مدينة الملك عبد العزيز للعلوم والتقنية
المنظمة العربية للترجمة

أندريا دي لوتشيا، فيلومينا فيروتشي
جيني تورتورا، ماريزيو توتشي

المنهجيات والتقنيّات وإدارة العمليات الحديثة في هندسة البرمجيّات

ترجمة

مرفت سلمان

سلسلة كتب التقنيات الاستراتيجية والمتقدّمة

المنهجيات والتقنيّات
وإدارة العمليات الحديثة
في هندسة البرمجيّات

المنظمة العربية للترجمة

أندريا دي لوتشيا، فيلومينا فيروتشي
جيني تورتورا، ماريزيو توتشي

المنهجيات والتقنيّات وإدارة العمليات الحديثة في هندسة البرمجيّات

ترجمة

مرفت سلمان

مراجعة

محمد الهيري

توزيع: مركز دراسات الوحدة العربية

الفهرسة أثناء النشر - إعداد المنظمة العربية للترجمة
المنهجيات والتقنيّات وإدارة العمليات الحديثة في هندسة البرمجيّات/ تحرير أندريا دي لوتشيا [وآخرون]؛ ترجمة مرفت سلمان؛ مراجعة محمد الهدى.
432 ص. - (تقنيات استراتيجية ومتقدمة - تقنية المعلومات؛ 2)
يشتمل على فهرس.
ISBN 978-9953-82-384-3
1. الهندسة الإلكترونية. 2. البرمجة الآلية. أ. دي لوتشيا، أندريا (تحرير). ب. سلمان، مرفت (مترجمة). ج. الهدى، محمد (مراجع). د. السلسلة.
005.1

«الآراء الواردة في هذا الكتاب لا تعبّر بالضرورة
عن اتجاهات تتبناها المنظمة العربية للترجمة»

Emerging Methods, Technologies,
and Process Management in Software Engineering
Edited by Andrea De Lucia [et al.]

المنظمة العربية للترجمة
بناية «بيت النهضة»، شارع البصرة، ص. ب: 5996 - 113
الحمراء - بيروت 2090 1103 - لبنان
هاتف: 753031 - 753024 (9611) / فاكس: 753032 (9611)
e-mail: info@aot.org.lb - http://www.aot.org.lb

توزيع: مركز دراسات الوحدة العربية
بناية «بيت النهضة»، شارع البصرة، ص. ب: 6001 - 113
الحمراء - بيروت 2407 2034 - لبنان
تلفون: 750084 - 750085 - 750086 (9611)
برقياً: «مرعربي» - بيروت / فاكس: 750088 (9611)
e-mail: info@caus.org.lb - Web Site: http://www.caus.org.lb

الطبعة الأولى: بيروت، آذار (مارس) 2011

المحتويـــات

الجزء الأول
هيكليات البرمجيات

الجزء الثاني
الأساليب الحديثة

الجزء الثالث
تقنيات تطوّر البرمجيات

الجـزء الرابـع
إدارة العمليـات

تقديـم

سلسلة كتب التقنيات الاستراتيجية
مبادرة الملك عبد الله للمحتوى العربي

يطيب لي أن أقدم لهذه السلسلة التي جرى انتقاؤها في مجالات تقنية ذات أولوية للقارئ العربي في عصر أصبحت فيه المعرفة محركاً أساسياً للنمو الاقتصادي والتقني، ويأتي نشر هذه السلسلة بالتعاون بين مدينة الملك عبد العزيز للعلوم والتقنية والمنظمة العربية للترجمة، ويقع في إطار تلبية عدد من السياسات والتوصيات التي تعنى باللغة العربية والعلوم، ومنها:

أولاً: البيان الختامي لمؤتمر القمة العربي المنعقد في الرياض 1428هـ 2007م الذي يؤكد ضرورة الاهتمام باللغة العربية، وأن تكون هي لغة البحث العلمي والمعاملات حيث نص على ما يلي: (وجوب حضور اللغة العربية في جميع الميادين، بما في ذلك وسائل الاتصال، والإعلام، والإنترنت وغيرها).

ثانياً: «السياسة الوطنية للعلوم والتقنية» في المملكة العربية السعودية التي انبثق عنها اعتماد إحدى عشرة تقنية إستراتيجية هي: المياه، والبترول والغاز، والبتروكيميائيات، والتقنيات المتناهية الصغر (النانو)، والتقنية الحيوية، وتقنية المعلومات، والإلكترونيات والاتصالات والضوئيات، والفضاء والطيران، والطاقة، والمواد المتقدمة، والبيئة.

ثالثاً: مبادرة الملك عبد الله للمحتوى العربي التي تفعّل أيضاً ما جاء في البند أولاً عن حضور اللغة العربية في الإنترنت، حيث تهدف إلى إثراء المحتوى العربي عبر عدد من المشاريع التي تنفذها مدينة الملك عبد العزيز للعلوم والتقنية بالتعاون مع جهات مختلفة داخل المملكة وخارجها. ومن هذه المشاريع ما يتعلق برقمنة المحتوى العربي القائم على شكل ورقي وإتاحته على

شبكة الإنترنت، ومنها ما يتعلق بترجمة الكتب الهامة، وبخاصة العلمية، مما يساعد على إثراء المحتوى العلمي بالترجمة من اللغات الأخرى إلى اللغة العربية بهدف تزويد القارئ العربي بعلم نافع مفيد.

تشتمل السلسلة على ثلاثة كتب في كل من التقنيات التي حددتها «السياسة الوطنية للعلوم والتقنية». واختيرت الكتب بحيث يكون الأول مرجعاً عالمياً معروفاً في تلك التقنية، ويكون الثاني كتاباً جامعياً، والثالث كتاباً عاماً موجهاً إلى عامّة المهتمين، وقد يغطي ذلك كتاب واحد أو أكثر. وعليه، تشتمل سلسلة كتب التقنيات الاستراتيجية والمتقدمة على ما مجموعه ثلاثة وثلاثون كتاباً مترجماً، كما خصص كتاب إضافي منفرد للمصطلحات العلمية والتقنية المعتمدة في هذه السلسلة كمعجم للمصطلح.

ولقد جرى انتقاء الكتب وفق معايير منها أن يكون الكتاب من أمهات الكتب في تلك التقنية، ولمؤلفين يشهد لهم عالمياً، وأنه قد صدر بعد عام 2000، وأن لا يكون ضيِّق الاختصاص بحيث يخاطب فئة محدودة، وأن تكون النسخة التي يترجم عنها مكتوبة باللغة التي أُلّف بها الكتاب وليست مترجمة عن لغة أخرى، وأخيراً أن يكون موضوع الكتاب ونهجه عملياً تطبيقياً يصبّ في جهود نقل التقنية والابتكار، ويساهم في عملية التنمية الاقتصادية من خلال زيادة المحتوى المعرفي العربي.

إن مدينة الملك عبد العزيز للعلوم والتقنية سعيدة بصدور هذه المجموعة من الكتب، وأود أن أشكر المنظمة العربية للترجمة على الجهود التي بذلتها لتحقيق الجودة العالية في الترجمة والمراجعة والتحرير والإخراج، وعلى حسن انتقائها للمترجمين المتخصصين، وعلى سرعة الإنجاز، كما أشكر اللجنة العلمية للمجموعة التي أنيط بها الإشراف على إنجازها في المنظمة وكذلك زملائي في مدينة الملك عبد العزيز للعلوم والتقنية الذين يتابعون تنفيذ مبادرة الملك عبد الله للمحتوى العربي.

الرياض 20/ 3/ 1431 هـ
رئيس مدينة الملك عبد العزيز للعلوم والتقنية
د. محمد بن إبراهيم السويل

مقدمــة

«استثمر وقتك في تطوير نفسك من خلال الاطلاع على ما كتبه الآخرون، بحيث تكتسب بسهولة خبرات ومعارف ما عانى وتعب الآخرون في تحصيله».

سقـراط

تؤثر البرمجيات في حياتنا اليومية بشكل كبير، وتعم مجتمعاتنا المدنية باستمرار. توفّر الابتكارات المتواصلة في تكنولوجيا المعلومات والاتصالات فرصاً جديدة وتزيد مدى التطبيقات التي يمكن الحصول عليها. تشكّل هذه الاختراعات تحديات تقنية وعملية جديدة لمهندسي البرمجيات الذين يجدون الحلول لاستثمار التكنولوجيا الجديدة بصورة أفضل وتطوير أنواع جديدة من التطبيقات البرمجية. في الوقت ذاته، يجب على مهندسي البرمجيات صيانة النظم البرمجية المستخدمة حالياً وتطويرها حتى تصبح متوافقة مع التغيرات المطلوبة على المتطلبات، إضافة إلى استثمار الفرص التي توفرها التكنولوجيا الجديدة بأكمل صورة. نتيجة لتطور البرمجيات، ازداد حجم ودرجة تعقيد النظم البرمجية بصورة مطّردة، ما أدى إلى زيادة التحديات التي تواجه عمليات تطوير البرمجيات وصيانتها أكثر وأكثر من ذي قبل، بحيث يتطلب ذلك وسائل محسّنة.

إذاً يمكننا القول إن **هندسة البرمجيات هي نظام ديناميكي** للغاية يجب أن يتطور باستمرار عن طريق البحث عن وسائل وأدوات ومنهجيات جديدة حتى تصبح عمليات تطوير البرمجيات وصيانتها أكثر موثوقية وكفاءة. يجب أن تؤخذ التبادلات الحرجة المرتبطة بالتكاليف والجودة والمرونة بالحسبان. لهذا السبب، تُعتبر هندسة البرمجيات اليوم إحدى مجالات البحث الأكثر متعة وتحفيزاً والأكثر ربحاً، كما إن لها تأثيرات عملية مهمة على صناعة البرمجيات.

يلقي هذا الكتاب الضوء على بعض آخر المستجدات في حقل هندسة البرمجيات. تتكوّن فصول الكتاب من بعض المحاضرات التعليمية التي حاضر فيها عدد من روّاد البحث المعروفين دولياً في أول ثلاث محاضرات في الندوة الدولية الصيفية التي نظمتها جامعة ساليرنو في إيطاليا حول هندسة البرمجيات.

لا يهدف هذا الكتاب إلى شمول جميع مواضيع هندسة البرمجيات، لكن يهدف إلى تقديم طائفة جيدة من البحوث الحالية في مجال هندسة البرمجيات التي تتعامل مع بعض المواضيع الأكثر ارتباطاً وتطوراً في المجتمع العلمي .**هذا الكتاب مخصص لطلاب الدراسات العليا** الذي يرغبون في التعمق في هذا الحقل، إضافة إلى أهميته للباحثين الذي يعملون في مجالات هندسة البرمجيات المختلفة.

تم تنظيم **هذا الإصدار في أربعة أجزاء: هيكليات البرمجية، الأساليب والتقنيات الحديثة في تطور البرمجية وإدارة العمليات.**

يتضمن **الجزء الأول**، في فصوله الثلاثة، تصميم هيكلية البرمجية، وهو نشاط حاسم مهم في تطوير النظم البرمجية. فهو يركّز على تأسيس الهيكلية الكلية للنظام البرمجي، وذلك بتعريف عناصر النظام الأساسية والترابط بينها. وهو يوفّر فكرة نظرية مهمة تساعد في فهم تعقّد النظام. في السنوات القليلة السابقة، تطورت هيكليات البرمجيات من كونها بُنى معرَّفة مسبقاً متراصة ومركزية لتصبح غير مركزية وموزعة، وتتكوّن من عناصر ومكوّنات متحدة بصورة ديناميكية.

يركّز **الفصل الأول** على هذا التطور من منظور تطور آليات تركيب البرمجيات، والمقصود هنا طريقة تركيب النظام ككل عن طريق ربط مكوّناته بعضها ببعض. يبيّن هذا الفصل كيفية تطور آليات الربط من خطط ثابتة إلى خطط ديناميكية ذات قابلية تغيّر عالية وفقاً للاكتشاف والتفاوض والتحسين. كما يتضمن الفصل الأول قائمة قصيرة من التحديات المهمة التي لابدّ أن تكون جزءاً من الأجندة المستقبلية لأبحاث هندسة البرمجيات.

يركّز **الفصل الثاني** على مجموعات المنتجات البرمجية التي حققت قبولاً واعتماداً واسعاً في صناعة النظم، وأصبحت جزءاً لا يتجزأ منه. لا تقوم العديد من الشركات بتطوير البرمجيات من نقطة الصفر، لكنها تركز على القواسم المشتركة بين المنتجات المختلفة وتضمّنها في هيكلية المنتج الذي تعمل على تطويره مع مجموعة من الموجودات ذات العلاقة التي يمكن إعادة استخدامها. يحدد هذا الفصل العديد من التحديات التي تشكّلها زيادة مدى خطوط المنتجات

الناجحة، وتقدّم مفاهيم جديدة للوصول إلى خطوط المنتجات البرمجية.

يركّز **الفصل الثالث** على حلول أنماط أو قوالب التصاميم التي عالجها المطوّرون عبر الوقت لحلّ طائفة من مشكلات التصميم المتكررة، ويعنون عمليات تطوير الأنماط كائنية التوجه التي يصعب فهمها وذلك بتوفير عرض عملي من خلال دراسة حالة. يتوفّر في هذا الفصل سلسلة من المتطلبات غير الوظيفية ذات درجة صعوبة متزايدة للعبة تفاعلية، كما تتوفر أنماط التصميم كالمراقب (Observer) والموائم (Adapter) والاستراتيجية (Strategy) والمصنع المجرد (Abstract Factory) بهدَف توفير حلول لهذه المتطلبات.

يشتمل **الجزء الثاني** على بعض الأساليب الحديثة كالأدوات البرمجية (Agents)، والحوسبة خدمية التوجه (Service-oriented computing)، واختبار النظم كائنية التوجه، والأساليب الرسمية (Formal methods)، وتطوير تطبيقات الويب (Web development). يناقش **الفصل الرابع** تأثير هندسة البرمجيات أدواتية التوجه (Agent Oriented Software Engineering - AOSE) في الحوسبة خدمية التوجه. النظام الأدواتي هو طريقة للتفكير في الأنظمة التي تتكون من كوائن نشطة وأدوات وسلوكها الجماعي. إن المجاز في الأداة فاعل في بناء البرمجية التي ستعمل ضمن نُظم مرتبطة بشبكة معقّدة، حيث ينعدم التحكم الشامل. بشكل خاص، أخذ المؤلفون في الحسبان بعض الأفكار الرئيسة لمفهوم الهيكلية خدمية التوجه (SOA) في سياق التقنيات الخاصية بالأدوات وذلك لتنسيق خدمات الشبكة ومكوّناتها.

يبيّن **الفصل الخامس** مسألة جودة البرمجيات، ويركّز على الأساليب الحديثة لاختبار النُظم كائنية التوجه. اختبار البرمجيات نشاط حاسم، وهو ضروري لضمان دقة الاعتمادية على البرمجية. وعلى الرغم من أن النموذج كائني التوجه يتيح تجنب بعض المشكلات المرتبطة بالبرمجة الإجرائية، فهو يعرّف بعض المشكلات التي لم تستطع أساليب الاختبار التقليدية تحديدها. في هذا الفصل، يقدم المؤلفون أحدث الحلول التي قدمتها الأبحاث الخاصة لاختبار النظم كائنية التوجه التي تتغلب على المشكلات الناتجة من بعض خصائص النُظم كائنية التوجه كالسلوك المعتمد على الحالة والتضمين والتوارث وتعدد الأشكال.

يوفّر **الفصل السادس** تقارير عن مشروع بيئات التصميم للتطبيقات الشاملة (DEGAS) الذي موّله الاتحاد الأوروبي، ويهدف هذا المشروع إلى دمج

استخدام لغة النمذجة الموحدة (UML) لتصميم التطبيقات الشاملة مع الأساليب الرسمية لتحليلها والتحقق منها. تتكون هذه الأنظمة الحديثة من أجهزة حاسبة موصولة من خلال شبكات الحاسوب. هذا وتعتبر عمليات التحليل والتحقق من أمن بروتوكولات الاتصال المستخدمة في مثل هذه الأنظمة الموزعة أمراً غاية في الأهمية. يستثمر هذا المشروع منهجاً يعتمد على عملية حسابية لتحديد النموذج السلوكي لأمن النظام الذي تم تنفيذ عمليات تحليل مكملات الجوانب البنيوية له على نماذج لغة النمذجة الموحّدة. يُلقي هذا الفصل الضوء على بعض المظاهر المرتبطة بلغة النمذجة الموحّدة التي تمثّل تدويناً تصويرياً معيارياً لتحديد وبناء وتوثيق بيانات النظام البرمجي. تمّ عرض إطار عملي يدعم مصمم النظام في تحديد البرتوكولات التي يجب تحليلها للوقوف على الخروقات التي قد تحدث في عمليات التحقق في نماذج لغة النمذجة الموحدة.

يراجع **الفصل السابع** الاتجاهات الحالية والمستقبلية في تطوير التطبيقات التي تعمل من خلال شبكة الإنترنت وتختبرها من وجهة نظر هندسة البرمجيات. لا شك أن شبكة الإنترنت هي إحدى أهم الاختراعات التي ظهرت في العقد الأخير من القرن الماضي، التي أثّرت في حياتنا اليومية بشكل كبير. إن الارتقاء المطّرد في تكنولوجيا المعلومات والاتصالات بيّن التطور السريع في شبكة الإنترنت، وفي الوقت نفسه وبصورة عكسية، فإن التطور السريع في الإنترنت قد بيّن الارتقاء المطّرد في تكنولوجيا المعلومات والاتصالات. كان عقد واحد من الزمان كافياً لنقل شبكة الإنترنت من كونها مجرد مستودع لعدد هائل من الصفحات المستخدمة للوصول إلى المعلومات الثابتة لتصبح منصّة فاعلة لتطوير وتشغيل ونشر العديد من التطبيقات. في الواقع، تتيح تقنيات الإنترنت ولغات البرمجة والمنهجيات الحديثة إنشاء تطبيقات ديناميكية توفر نموذجاً جديداً للتعاون والتشارك بين عدد هائل من المستخدمين. يركّز الفصل السابع تحديداً على التطورات التي طرأت على تقنيات التصفّح، وعلى خادم شبكة الإنترنت (Web Server)، وعلى البنى التحتية لشبكات الحاسوب، وعلى مستوى التطبيقات واتجاهات هندسة البرمجيات.

يتضمن **الجزء الثالث** التقنيات المستخدمة في تطور البرمجيات. إن الطبيعة الديناميكية للبرمجيات هي إحدى أهم مميزاتها، ليس ذلك فحسب، بل إنها إحدى أهم أسباب تعقيد البرمجية. في الواقع، تحتاج البرمجيات إلى تطور مستمر لتلبّي المتطلبات المتغيّرة في هذا العالم، ولتشمل التقنيات المبتكرة. أما

الفصل الثامن فيُعْنونِ الأمور التي قدمتها الهيكليات خدمية التوجه التي تمثّل أحدث التغيرات في التكنولوجيا محددة بذلك انفصالاً جذرياً عن التقنيات السابقة. يوضح هذا الفصل الاستراتيجيات المختلفة لتزويد الهيكلية خدمية التوجه بخدمات شبكة الإنترنت ويركّز على كيفية استعادة خدمات الإنترنت من التطبيقات المستخدمة حالياً. كما يلقي هذا الفصل الضوء على العديد من قضايا البحث التي تستحق التحقق منها في هذا السياق.

يلقي **الفصل التاسع** الضوء على مشكلة تحليل تطور النظم البرمجية للحصول على معلومات عن أسباب وتأثير التغيرات المعيّنة، وللحصول على صورة واضحة عن المشكلات المتعلقة بخصائص معيّنة. إن ذلك أمر حاسم في التعامل مع زيادة درجة تعقد وسوء الهيكلية، ويتطلب تقنيات فاعلة لنقل المعلومات ذات العلاقة التي توجد ضمن كمية هائلة من البيانات. يقدم هذا الفصل بعض التقنيات التصورية التي تتيح تحليل جوانب مختلفة للنظم البرمجية.

أما **الجزء الرابع** فيركّز على إدارة العمليات وهي موضوع آخر رئيس في هندسة البرمجيات. في الواقع، تم إدراك حقيقة أن جودة النظام البرمجي تتأثر بشكل كبير بجودة العملية المستخدمة لتطوير النظام وصيانته. إدارة العمليات تعني استخدام المهارات والمعرفة والأدوات والتقنيات لتعريف وقياس العمليات والسيطرة عليها وتحسينها، وذلك للتحقق من جودة النظم البرمجية بطريقة تحقق كفاءة من ناحية التكاليف. في هذا السياق، تؤدي الاختبارات التجريبية المبنية على الملاحظة دوراً مهماً في نقل نتائج أبحاث هندسة البرمجيات إلى عمليات صناعة البرمجيات ولتحويل الخبرات إلى معرفة. يقدم **الفصل العاشر** مفاهيم أساسية معروفة في الأدبيات، ثم يشرح الخطوات التي تنفذ في الأبحاث العلمية بدءاً من الملاحظة وحتى الخروج بنظرية، وكيفية دعم الأبحاث التجريبية لعمليات البرمجة والتطوير في هندسة البرمجيات.

أما **الفصل الحادي** عشر وهو الأخير في الكتاب فيعرض أسس المنهجيات السريعة ومجموعة تقنيات التطوير التي ظهرت خلال عقد التسعينيات من القرن الماضي، التي صمّمت لتحدد بعض المشكلات التي تعترض عملية تطوير البرمجيات الحديثة، كتسليم المنتج في الوقت المحدد وحسب الموازنة التي وضعت له وبجودة عالية تحقق متطلبات واحتياجات العميل، بما في ذلك منهجية (eXtreme Programming) (XP) وهي أكثر المنهجيات شهرة، وقد قدمها

كينت بيك (Kent Beck). تقدم هذه الأساليب بدائل ملائمة للمنتج الصحيح من دون هدر الوقت والجهد، في سياقات محددة ولمشكلات محددة. قام المؤلفون بتحليل المفاهيم الرئيسة لمنهجيات التطوير السريعة والصعوبات التي تعترض تطبيقها بفعالية وكفاءة، مركزين تحديداً على منهجية XP.

نقدّم امتناننا للكثير من الأشخاص الذين دعموا نشر هذا الإصدار مخصصين وقتهم وطاقتهم. بدايةً، نشكر جميع المؤلفين لمساهمتهم القيّمة. كما نقدّم شكرنا الجزيل لأعضاء اللجنة العلمية على عملهم ودعمهم للدورة الدولية حول هندسة البرمجيات. كما نقدّم شكرنا لدائرتنا (دائرة الرياضيات والمعلوماتية في جامعة ساليرنو) لدعمهم ومساعدتهم المتواصلة. كما إننا ممتنون لكلٍّ من فوستو فاسانو (Fausto Fasano)، روكو أوليفيتو (Rocco Oliveto)، سيرجيو دي مارتينو (Serjio Di Martino)، ريتا فرانسينس (Rita Francense)، غياسبي سكانيللو (Giuseppe Scanniello)، مونيكا سيبيللو (Monica Sebillo)، فينسينزو دوفيميا (Vincenzo Deufemia)، كارمن غرافينو (Carmine Gravino)، وميشيل ريسي (Michele Risi) الذين كانوا خير عون لنا في تنظيم الإصدارات المختلفة من الدورة. وأخيراً، نود أن نشكر دار وايلي للنشر لمنحنا فرصة نشر هذا الإصدار وجميع فريق العمل ذوي العلاقة، ونخص بالذكر المحررة كايسي كريغ (Cassie Craig) ومحرري المنتج ليزا فان هورن (Lisa Van Horn) من دار وايلي للنشر وباول بيني (Paul Beaney) وبرافينا باتيل (Pravina Patel) من دار تشيسيت للتأليف.

نأمل أن تستمتع عزيزنا القارئ بقراءة هذا الكتاب، وأن تجد فيه ما يفيدك في عملك. كما نتمنى أن نكون قد عالجنا المواضيع التي تساعدك في أبحاثك في هندسة البرمجيات، وفي مساهمتك معنا في الندوة العالمية لهندسة البرمجيات.

أندريا دي لوتشيا (Andrea De Lucia)

فيلومينا فيروتشي (Filomena Ferrucci)

جيني تورتورا (Genny Tortora)

ماريزيو توتشي (Maurizio Tucci)

كانون الأول/ ديسمبر 2007

الجزء الأول

هيكليات البرمجيات

1

نشوء آليات تركيب البرمجيات: دراسة

كارلو غيزي (Carlo Ghezzi)
وفيليبو باسيفيكي (Filippo Pacifici)

1 ـ 1 مقدمة

في الهندسة، يتضمن تطوير أي نظام تصميم هيكليته عن طريق تحليله إلى أجزاء منفصلة وتوضيح العلاقات بينها. تتيح هذه الطريقة للمهندسين الوقوف على درجة تعقيد النظام وذلك بتطبيق مبدأ «فرّق تسد» (Divide-and-Conquer) أي التركيز المتكرر على عدد محدود من الأنظمة الفرعية وعلى التفاعل ما بينها.

بناءً على ما تقدم، قد يكون ممكناً تحديد مرحلتين: تحديد سلوك الأجزاء منفصلة، وتحديد سلوك الأجزاء مجتمعة. إن تركيب هذه الأجزاء هو طريقة بناء النظام كاملاً عن طريق ربطها مع بعضها بعضاً. في هذه المرحلة، يركّز المهندسون على كيفية تأسيس العلاقات بين الأجزاء بدلاً من التركيز على تركيب داخلي محدد أو على سلوك المكوّنات.

تطورت النُظُم البرمجية من نظم صغيرة وموحّدة إلى نظم كبيرة وموزعة. بناء على ذلك، اكتسبت عملية تركيب النظم أهمية أكبر من ذي قبل نظراً إلى أن مهندسي النظم قد أدركوا أن تطوير البرمجيات لا يعتبر مهمة موحدة يضطلع بها شخص واحد، بل أن النظم البرمجية يجب أن توصف على أنها بنى معقدة يتم الحصول عليها من خلال التنفيذ الحذر لعدد من آليات التركيب المحددة.

يمكن تحليل تركيب البرمجية باتجاهين: المعالجة وهيكلية المنتج. من وجهة نظر المعالجة، يجب أن يتم التركيب حسب طريقة بناء البرنامج في ما يتعلق بوحدات العمل والتنظيمات. أما من وجهة نظر هيكلية المنتج، فالتركيب يعني طريقة بناء المنتجات في ما يتعلق بالمكوّنات وروابطها.

يركّز هذا الفصل على تطور مبادئ تركيب البرمجيات وآلياتها خلال العقود الماضية. لقد سارت عملية التطور باتساق في اتجاه زيادة المرونة: من مكوّنات ثابتة (Static) إلى مكوّنات متغيرة (Dynamic) ومن مكوّنات برمجية مركزية إلى مكوّنات وعناصر برمجية غير مركزية. على سبيل المثال، لقد حدث تطور على مستوى شيفرة البرمجة من تراكيب برنامج موحد إلى تحليلات وظيفية وبرمجية كائنية التوجه. أما على مستوى أعلى ونظري أكثر فتطورت هيكليات البرمجيات من مركبات هيكلية مقترنة بشدة، كما في حالة الهيكليات متعددة الطبقات (Mult-Tier) إلى مكوّنات غير مركزية من نوع نظير ـ نظير. كما تطورت عمليات البرمجيات من عمليات ثابتة ومتعاقبة وموحدة إلى مخططات سير عمل سريعة وذات فترات منتظمة تكرارية وغير مركزية[9].

يبحث هذا الفصل في عدد من مفاهيم هندسة البرمجيات ويضعها في منظور تاريخي. يمكن اعتبار هذا الفصل بمثابة منهج تدريبي عن مكوّنات البرمجيات.

تم تنظيم هذا الفصل كما يأتي. يتضمن القسم 1-2 بعض المفاهيم الأساسية عن مكوّنات البرمجيات. أما القسمان 1-3 و1-4 فيناقشان متى تظهر مشكلة مكوّنات البرمجيات، وكيف يمكن معالجتها مبدئياً. يركّز القسم 1-5 على بعض المساهمات العملية حول المفاهيم والأساليب والتقنيات التي تدعم تصميم تطور البرمجيات. يدور القسم 1-6 حول المرحلة الحالية، وهو يركّز على أن التحدي الأساسي هو كتابة التطور البرمجي المتواجد في العالم المفتوح. فهو يعرف أين تكمن التحديات الأساسية، ويشرح بعض اتجاهات البحث الممكنة.

1 ـ 2 مفاهيم أساسية

إن مفهوم الربط (Binding)[21] هو أمر أساسي دائم التكرار لفهم تركيب البرمجية على مستوى الشيفرة البرمجية والهيكلية. الربط هو تأسيس العلاقة بين العناصر التي تكون هيكلية البرمجية. أما زمن الربط فهو الزمن الذي يتم عنده

تأسيس هذه العلاقة. على سبيل المثال، قد تستخدم هذه المفاهيم لوصف الخصائص الدلالية للغات البرمجة التي قد تختلف في السياسة المتبناة لربط المتغيرات مع أنواعها وربط الأنواع الفرعية (Subclasses) بالأنواع الأم (Classes) التي تتبع لها، وربط تنفيذ الوظائف بتعريفاتها أو ربط الشيفرة البرمجية القابلة للتنفيذ مع الأجهزة الافتراضية. أما على المستوى الهيكلي، فقد يتم ربط الخادم (Server) بالعديد من الأجهزة التابعة (Clients). أزمان الربط النموذجية هي: فترة التصميم وفترة الترجمة وفترة النشر (Deployment) وفترة التنفيذ (Run Time). ربط فترة التصميم هو، مثلاً، اتحاد يربط النوع في مخطط النوع في لغة النمذجة الموحدة مع النوع الأم. أما ربط فترة الترجمة فيتم عند معالجة وصف الشيفرة المصدرية[1].

بطريقة مماثلة، ثمة حالات تُنفذ فيها عملية الربط بين وحدة معيّنة في البرنامج قابلة للتنفيذ والنقطة التي يتم عندها التنفيذ في النظام الموزع عند فترة النشر. وفي حالات أخرى، يتم تنفيذ الربط عند زمن التنفيذ وذلك لدعم إعادة تهيئة النظام.

ومن المفاهيم المتعامدة الأخرى ما يعرف بثبات الربط. يكون الربط ثابتاً (Static) إذا لم يتغير بعد تأسيسه، وبخلاف ذلك، يكون ديناميكياً (Dynamic). في معظم الحالات، يكون ربط زمن التصميم وزمن الترجمة وفترة النشر ثابتاً، بينما يكون ربط زمن التنفيذ متغيراً، رغم ذلك، هناك بعض الاستثناءات. يكون الاختلاف الرئيس بين ربط زمن ما قبل التنفيذ، وزمن التنفيذ: فربط زمن التنفيذ يضيف مرونة أكثر إلى النظام في ما يتعلق بربط زمن ما قبل التنفيذ وذلك بسبب سياسات التغيير التي يمكن أن تتبع لها عملية الربط. في حالة الربط الديناميكي، قد نفرق أكثر بين الربط الواضح والربط الأوتوماتيكي. في إحدى الحالات، مثلاً، يطلب المُستخدم من النظام تغيير الربط، ويحدد عنصراً جديداً يجب ربطه، بينما في حالة أخرى يكون للنظام استقلالية في تحديد متى يجب تغيير الربط، ويكون في هذه الحالة مزوّداً بآليات لتحديد العنصر الذي يجب ربطه. عملياً، هناك طائفة من الحلول

---

(1) نستخدم هذا المصطلح بدلاً من المصطلح الأكثر شيوعاً «فترة التجميع compile-time» وهو مصطلح أكثر حصراً، وذلك لشمول مظاهر معالجة في لغات برمجة أخرى كالمعالجة المسبقة (preprocessing) وربط/ تحميل.

الممكنة التي تتراوح بين ربط زمن التنفيذ الديناميكي الكامل وربط زمن ما قبل التنفيذ الثابت.

هذا وقد تنطبق هذه المفاهيم على مستوى العملية أيضاً لوصف وتحديد كيفية تنظيم عملية تطوير البرمجية. على سبيل المثال، قد يربط أحدهم بين الأشخاص وأدوار وظيفية محددة (كمختبِر النظام) بطريقة ثابتة، أو قد يتغير ذلك في أثناء تنفيذ عملية التطوير. بطريقة مماثلة لذلك، قد يتطلب إكمال مرحلة التطوير (كتفصيل التصميم مثلاً) قبل البدء بالمرحلة التالية (كتابة الشيفرة البرمجية). في هذه الحالة، يتم تأسيس الربط بين مرحلة ما والمرحلة التي تليها من خلال مخطط تحكم متتالي معرف مسبقاً. وقد يقوم أحدهم بتنظيم المرحلتين كخطوتين متزامنتين مفصلتين.

إن وصف الربط مفيد لفهم تطور البرمجيات، سواء تم بالطريقة التي يتّبعها مهندسو النظم لتطوير البرمجيات أو بالطريقة التي يبنون بها هيكلية التطبيقات. لقد أدى هذا التطور إلى تحوّل النظم البرمجية من بنى ثابتة ومركزية وموحدة حيث يعرف الربط في زمن التصميم، بينما يكون مجمداً في أثناء دورة حياة النظام كلها، إلى تصاميم ديناميكية وقياسية وغير مركزية وإلى عمليات تصميم، حيث يمكن تغيير الربط الذي يعرّف بنى مقترنة على نحو غير محكم من دون توقف النظام، وقد يتخطى الحدود المشتركة بين الشركات.

ومن الجدير بالاهتمام محاولة تحديد مصادر التطور والقوى المحركة له. إن التقدم الذي طرأ في تكنولوجيا البرمجيات هو قوى ذاتية بالطبع. في الواقع، توفّر لغات البرمجة وأساليب التصميم الحديثة دعماً للربط الديناميكي والتغيّر المستمر. على كل حال، نشأت الحاجة إلى الديناميكية والتطور أساساً في بيئات العمل التي تتواجد ضمنها النظم البرمجية. أما الحدود بين هذه النظم البرمجية والبيئة الخارجية فهي عرضة للتغيير المستمر. لقد استخدمنا مصطلح «برمجيات العالم المفتوح» لوصف هذا المفهوم[7]. في بدايات تطوير البرمجيات، وضع افتراض حصر البرمجيات في عالم مغلق. لقد افترض أن المتطلبات التي تحدد الحدود بين البيئة والنظام الذي سيتم تطويره كانت ثابتة بما فيه الكفاية ويمكن انتقاؤها مقدماً قبل بدء التصميم والبرمجة. بما إننا سنفسر ذلك في ما تبقّى من هذا الفصل، أثبت هذا الافتراض خطأه. فالبرمجيات تتواجد في عالم مفتوح بحيث تتغير الحدود في العالم الفعلي بشكل مستمر، حتى في أثناء كون

البرمجيات قيد الاستخدام. يتطلب ذلك تغيير السياسات الإدارية التي يمكن أن تدعم التطور خلال روابط فترة التنفيذ.

1 ـ 3 البدايات

حتى عقد الستينيات من القرن الماضي[2]، كانت عملية تطوير البرمجيات مهمة يختص بها شخص واحد فحسب، كان على هذا الشخص فهم المسألة التي عليه حلّها فهماً جيداً، ومن ثم كان المبرمج هو الشخص المتوقع كمستخدم للتطبيق الناتج. وبالتالي، لم يكن ثمة تمييز بين المبرمجين والمستخدمين. كانت المشكلات التي توجب حلّها أساساً ذات طبيعية رياضية، وقد توفرت للمبرمجين معرفة في الرياضيات ما مكّنهم من فهم المسائل التي طُلب منهم حلها. كانت البرامج بسيطة نسبياً مقارنة بما هو متوفّر في وقتنا الحاضر، كما تمّت برمجة تلك البرامج بواسطة لغات برمجة ذات مستوى منخفض، ولم يكن هناك دعم للأدوات. أما بالنسبة إلى المعالجة، فلم يتّبع مهندسو البرمجيات أي نموذج تطوير نظامي، لكنهم اتّبعوا طريقة كتابة الشيفرة البرمجية وإصلاحها باستمرار، أي فترة برمجية تكرارية من كتابة الشيفرة وإصلاحها لتقليل الأخطاء.

لقد أثبتت هذه الطريقة عدم جدواها، ذلك بسبب (أ) زيادة تعقد النظم البرمجية و(ب) الحاجة إلى تطبيق الحوسبة، ليس في حلّ المسائل العملية فقط، بل أيضاً في حلّ المسائل المرتبطة بمجالات أخرى، كإدارة الأعمال ومراقبة العمليات، حيث يكون فهم هذه المشكلات أقل من لو أنها كانت علمية.

في بداية عقد السبعينيات من القرن الماضي، وبعد أن أصبح هناك إدراك لهندسة البرمجيات كموضوع علمي حاسم[31, 8]، تم تطوير منهجية البرمجة التي تُعرف بالشلال (Waterfall)[38] كنموذج عمليات مرجعي لمهندسي البرمجيات. كانت تلك محاولة لدمج مفهوم الانضباط وإمكانية التوقع في هندسة البرمجيات. تنفذ هذه المنهجية على شكل مخطط سير عمل ذي مراحل متعاقبة تبدأ بمرحلة تحديد المتطلبات تتبعها مرحلة التصميم، ثم مرحلة كتابة الشيفرة البرمجية، وأخيراً تكون مرحلة التحقق من فاعلية البرنامج هي المرحلة الأخيرة. كان نموذج المعالجة المقترح ثابتاً ولا يتغير: فقد كانت عملية التقسيم إلى

(2) لمطالعة موجز عن تاريخ هندسة البرمجيات، يمكن للقارئ الرجوع إلى المرجع 22.

مراحل وربطها في مرحلة واحدة محددة بدقة. كما إن عملية إنهاء مرحلة والخروج منها والدخول في المرحلة التالية مُعدّة بدقة، حيث كان الهدف الحد من إعادة أي عمل في أي من المراحل المكتملة أو حتى إن كان ذلك غير مسموح به. كان الدافع الأساسي هو أن إعادة العمل - أي التغيرات التي تطلب لاحقاً على البرمجية - سيكون على حساب الجودة، وسيتسبب في ارتفاع تكاليف البرمجة والتطوير، عدا عن أنه سيسبب تأخيراً في طرح المُنتَج في السوق. أدى ذلك إلى استثمار الجهود في عمليات تحديد المتطلبات والخصائص وتحليلها، التي يفترض أن تتوقف قبل البدء في البرمجة.

ساعدت هذه المنهجية في وضع افتراضات ضمنية قوية جداً على المجتمع الذي سيتم تشغيل البرمجية فيه. لقد افترض أن هذا المجتمع ثابت غير متغير، بمعنى أنه افترض أن متطلبات النظام لن تتغير. لقد كانت المشكلة في اختيار المتطلبات بشكل صحيح، بحيث تكون محددة تماماً قبل البرمجة. المتطلبات المحددة بصورة كاملة ودقيقة تضمن عدم ظهور أي حاجة إلى إجراء أي تغيير في البرمجية في أثناء عملية البرمجية أو بعد تسليم البرنامج أو تسويقه.

من الافتراضات الأخرى أن المؤسسات التي سيُستخدم فيها البرنامج ذات بنية موحدة. لقد كانت تلك المؤسسات عبارة عن وحدات منفصلة ذات إدارة مركزية. وقد كان التواصل والتنسيق بين الشركات الموحدة محدوداً وغير حاسم للأعمال التي تقوم بها. لذا فقد كانت الحلول المركزية أمراً طبيعياً في ظل هذه الظروف.

بناءً على هذه الافتراضات، وعلى تقنيات تطوير البرمجيات التي كانت المتاحة في ذلك الوقت، طوّر المهندسون نُظماً برمجية موحّدة ليتم تشغيلها من خلال أجهزة الحاسوب المركزية (Mainframe). ومع أن حجم البرامج تطلب مشاركة العديد من الأشخاص في جهود البرمجة إلا أنه أعطي القليل من الاهتمام بالهيكلية التي ترتكز على تقسيم البرنامج إلى وحدات. أما عملية تطوير الوظائف بشكل منفصل فقد كانت مقيدة بفترة الترجمة، ولم تكن المميزات التي تضاف للبرنامج متاحة لتدعم إجراء تغيير في البرنامج قيد الاستخدام إن لزم الأمر. لذا، فإنه لإجراء تغيير، كان الأمر يتطلب وضع البرنامج في حالة عدم استخدام، إلى أن يتسنى للمُبرمجين تعديل الشيفرة البرمجية، ومن ثم إعادة تجميع الوحدات وربطها وإعادة نشرها للاستخدام. إذاً، ونظراً إلى عدم

الاهتمام بالهيكلية التي ترتكز على تقسيم البرنامج إلى وحدات، كان من الصعب إجراء التغيير المطلوب، كما كان من الصعب توقع تأثير هذه التغيرات.

1 ـ 4 اكتساب المرونة

إن الحاجة إلى إنجاز التغيرات التي تطلب على البرمجيات والتطور الذي طرأ على هندسة البرمجيات دعت إلى البحث عن منهجيات أكثر مرونة. أصبح من الواضح أن متطلبات النظام لا يمكن أن يتم جمعها وتحديدها مقدماً في معظم الظروف والحالات وذلك لأن العملاء لا يعرفون ما يتطلبونه بالضبط مقدماً. حتى لو عرفوا متطلباتهم، فإن هذه المتطلبات عرضة للتغيير، قد يكون ذلك قبل برمجة البرنامج بالكامل. يطرح هذا الأمر تساؤلاً حول التوقعات الضمنية التي يرتكز عليها نموذج الشلال. إضافة إلى ذلك، تحولت الهيكليات البرمجية لتصبح صعبة التعديل نظراً إلى أن الاقترانات المحكمة بين أجزاء البرمجية المختلفة من المستحيل فصل تأثير التغيرات في الأجزاء المقيّدة من البرنامج. ظاهرياً التغيرات البسيطة لها تأثير شامل في تكامل النظام ككل.

أخيراً، أدّى التطور في مؤسسات الأعمال إلى الحاجة إلى مزيد من المرونة. لقد تطورت بنى المؤسسات المركزية الموحدة لتصبح وحدات ديناميكية مستقلة وغير مركزية. كما تغيرت عمليات تطوير البرمجيات أيضاً لأن المكوّنات الجاهزة للاستخدام وأطر العمل المختلفة أصبحت متوفّرة تدريجياً. أصبحت عمليات تطوير البرمجيات غير مركزية وموزعة على عدة مؤسسات، فهناك مطوّرو عناصر البرامج، وهناك من يقوم بدمج وحدات النظام.

سنناقش في الأجزاء الأخرى من هذا الفصل أهم الخطوات والمساهمات التي تقود إلى الحصول على مستويات متزايدة من المرونة.

1 ـ 4 ـ 1 تصميم التغيير

المساهمة المفاهيمية الأولى نحو تطور النظم البرمجية كانت مبدأ تصميم التغير وتعريف التقنيات التي تدعمه. اقترح بارناس (Parnas)[37-35] أنه يتوجب على مصممي النظم الاهتمام أكثر بمرحلة المتطلبات لفهم التغيرات التي قد تطرأ مستقبلاً على النظام. إذا كان بالإمكان توقع التغيير فإن التصميم يجب أن يحاول تحقيق هيكلية تضمن سهولة فصل تأثير التغيير ضمن أجزاء مقيّدة من

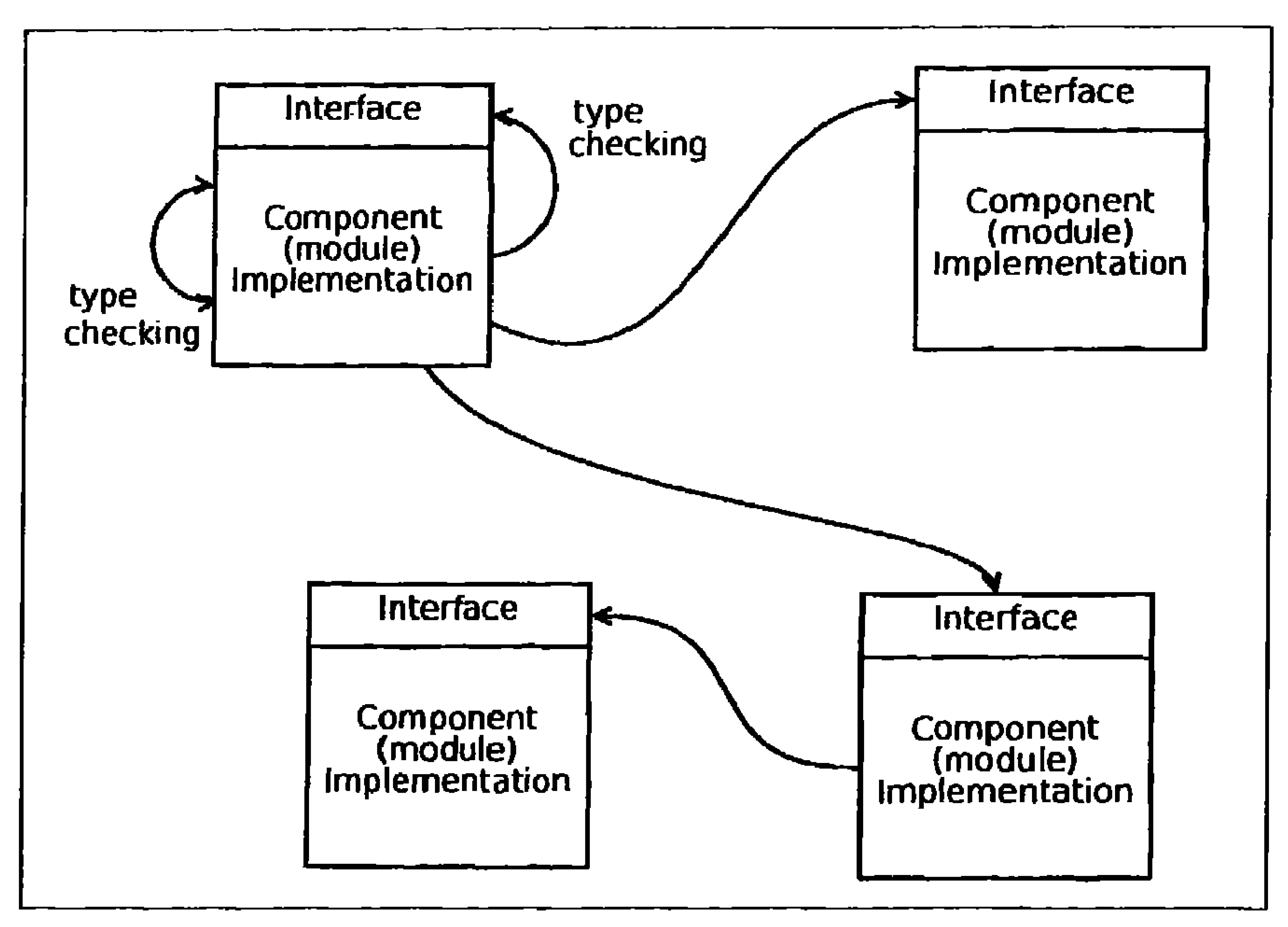
النظام. ومن أنواع التغيير التي يمكن أخذها بالحسبان مسبقاً: تغيير في الخوارزميات الحسابية المستخدمة لتنفيذ مهمة معيّنة، تغيير في عرض البيانات، تغيير في أدوات أو أجهزة معيّنة يستخدمها البرنامج ويتفاعل معها كالمجسّات والمحركات الميكانيكية.

يتوسع بارناس في تقنية تصميم تُعرف بإخفاء المعلومات، ومن خلالها يتم تحليل البرنامج إلى وحدات، حيث يتم إخفاء معظم مصادر التغيرات المتوقعة على المتطلبات كأسرار في الوحدة، ولا يكون الوصول إليها من قبل الوحدات الأخرى متاحاً.

ثمة فصل واضح بين واجهة الوحدة وتنفيذ الوحدة، يعمل على فك قرارات التصميم الثابتة المتعلقة باستخدام الوحدة من قبل وحدات أخرى من الأجزاء القابلة للتغيير والمختفية ضمنها. لا يتأثر مستخدمو الوحدة بالتغييرات طالما أنها لا تؤثر في واجهة الاستخدام. تتيح هذه الخاصية من خواص مبدأ التصميم الفصل بين القضايا، وهذا مبدأ يدعم التصميم متعدد المستخدمين ويدعم تطور البرمجيات.

يتيح إخفاء المعلومات تحليل تصميم النظم الكبيرة إلى وحدات متسلسلة، يكون لكل وحدة منها واجهة منفصلة عن التنفيذ. إذا تم تأسيس الربط بين الواجهة والتنفيذ بطريقة ثابتة ضمن فترة الترجمة فإنه يمكن التحقق بطريقة ثابتة من اتساق الواجهات مقابل التنفيذ ومقابل الاستخدام من قبل وحدات العميل. إن التحقق الثابت من شأنه أن يمنع بقاء الأخطاء غير معروفة في النظم بعد استخدامها.

إن مفاهيم إخفاء المعلومات والواجهة مقابل التنفيذ أدمجت في لغات البرمجة، وقد دعمت التطوير المنفصل والتجميع المنفصل للوحدات في النظم الكبيرة[39]. مثال على ذلك، دعنا نأخذ بالاعتبار لغة البرمجة (Ada) التي صمّمت في نهايات عقد السبعينيات من القرن الماضي. الوحدات في لغة البرمجة هذه[21] تعرف بالحزم، وهي تدعم إخفاء المعلومات، وفصل تجميع الواجهات والتنفيذات. فحالما يتم تحديد واجهة وتجميعها، فإنه يكون بالإمكان تجميع تنفيذ الواجهة وتنفيذ وحدات المستخدمين. عموماً، فإنه يمكن تجميع وحدة إذا كانت جميع الوحدات التي تعتمد عليها مجمعة أصلاً. وهذا يتيح للمجمع أن ينفذ تحققاً من النوع الثابت بين الوحدات المختلفة (انظر الشكل 1 ـ 1).



الشكل (1 ـ 1): هيكلية برنامج مجزأ إلى وحدات

هناك الكثير من الأمثلة على لغات برمجة توفّر خصائص مشابهة، ولو بشكل جزئي. على سبيل المثال، في لغة البرمجة *C* من الممكن فصل تعريفات النماذج الأولية للوظائف عن التنفيذ، ووضعها في ملفات مختلفة (عن ملفات التنفيذ)، لكن ذلك ليس بالأمر المتطلب. بهذه الطريقة، من الممكن تعريف الواجهة المصدرة من قبل وحدة، بالرغم من أنه لا يمكن تحديد الوظائف التي تستوردها الوحدة من وحدة أخرى بالضبط.

مع أن التغييرات قد سهّلت مفاهيمياً إذا كانت مفصولة ضمن تنفيذات الوحدة، فإن هيكلية التنفيذ الناتجة ستكون ثابتة. فقد تم حل جميع الروابط قبل فترة التنفيذ ولا يمكن تغييرها ديناميكياً. لإجراء تغيير، يجب أن يُعاد البرنامج كله إلى فترة التصميم، بحيث تُجرى التعديلات ويتم التحقق منها من قبل عملية الترجمة، وبعد ذلك يعاد تشغيل النظام ضمن تهيئة جديدة.

1 ـ 4 ـ 2 لغات البرمجة كائنية التوجه

كانت الطريقة المباشرة لإخفاء المعلومات مضمّنة في لغة وحدات وفّرت دعماً محدوداً لتطور البرمجيات والتركيب المرن للنظم البرمجية. وفّرت تقنيات

التصميم كائنية التوجه ولغات البرمجة خطوة كبيرة في هذا الاتجاه. إذ يهدف التوجه الكائني إلى توفير دعم للوحدات المستقلة والتطوير التزايدي؛ كما إنه يدعم التغيير الديناميكي. بناء على التصميم كائني التوجه، يتم تحليل النظام البرمجي إلى أنواع (Classes) بحيث يتم تجهيز البيانات والعمليات، وتعرف أنواع نظرية جديدة من البيانات يتم توريد عملياتها من خلال واجهة النوع.

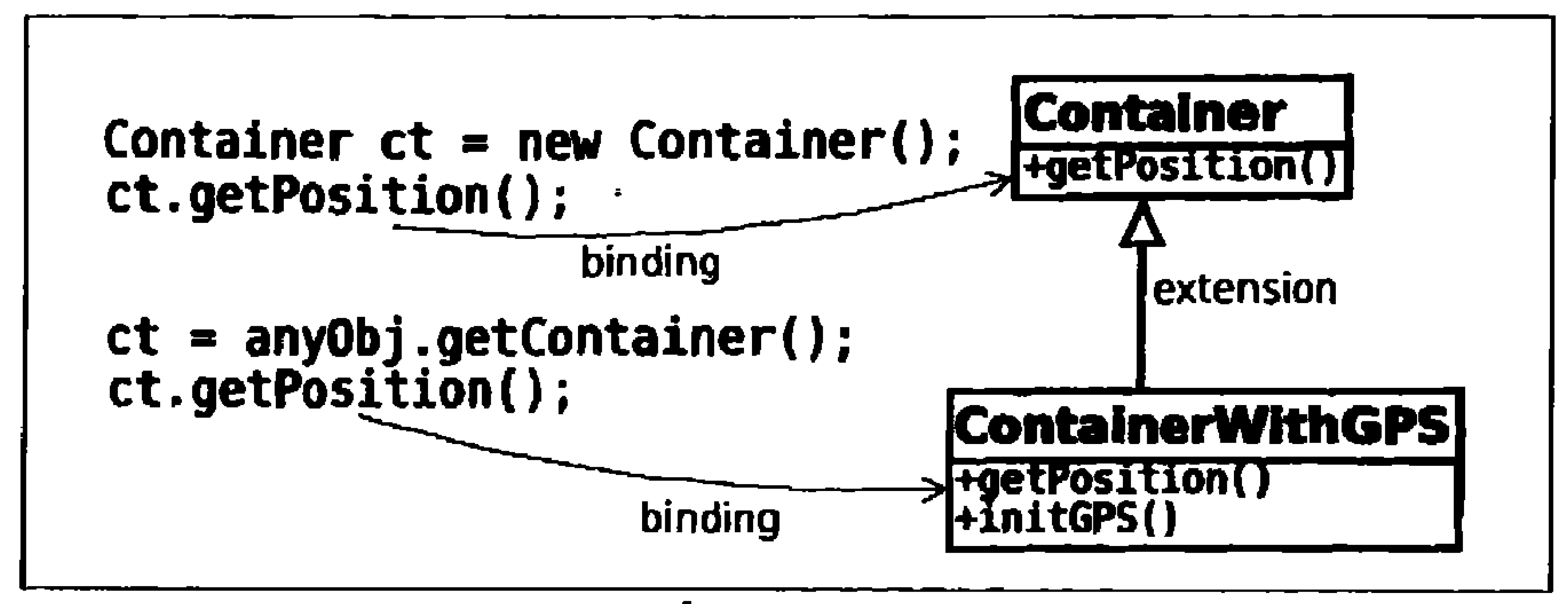
من المساهمات الكبرى للغات البرمجة كائنية التوجه أن الربط بين عملية تتطلبها وحدة عمل تابع (Client) والتنفيذ الذي توفره وحدة خادم (Server) قد يتغير ديناميكياً في أثناء فترة التنفيذ. وهذه خاصية أساسية لدعم تطور البرمجية بطريقة مرنة. إضافة إلى ذلك، توفر معظم لغات البرمجة كائنية التوجه هذه الخاصية بطريقة تحافظ على أمن عملية التحقق من النوع الثابت[21].

مثال على ذلك، دعنا نفترض وجود برنامج يُستخدم في مجال الإمدادات لمتابعة حاويات المواد. تتيح العمليات التي تختص بالحاويات للمستخدمين تحميل مادة إلى الحاوية والحصول على قائمة بالمواد التي تحويها وربط الحاوية بالوسيلة الناقلة التي قد تكون عبارة عن قطار أو شاحنة تحمل عليها الحاوية، أو قد تكون عبارة عن مساحة اصطفاف تخزن فيها الحاوية مؤقتاً. ثمة عملية تستخدم للحصول على موقع الحاوية الجغرافي ويتم تنفيذها عن طريق الطلب من الحاوية تحديد ذلك. موقع الاصطفاف ذو موقع ثابت، بينما يوجد وسائل معيّنة ليحدد كل من الشاحنة والقطار موقعهما ديناميكياً. لنفترض حدوث تطور على النظام بحيث تزود الحاويات بهوائي لنظام تحديد المواقع يمكن استخدامه لتحديد موقعها. يمكن تطبيق ذلك كتغير في تنفيذ العملية (get_position) التي تستخدم حالياً البيانات التي يوفرها نظام تحديد المواقع بدلاً من طلب ذلك من الناقل. هذا ولن يتأثر المستخدم الذي يستخدم الحاوية بطريقة تنفيذ العملية الجديدة، إذ يتم توجيه تنفيذ العملية (get_position) إلى عملية معادة التعريف (الشكل 1-2).

بشكل عام، إذا أعطينا نوعاً يحدد طرازاً نظرياً من البيانات، فمن الممكن تحديد التغيير بواسطة الأنواع الفرعية (المشتقة من النوع الرئيس). يرتبط النوع الفرعي بشكل ثابت مع النوع الرئيس (النوع الأم) من خلال علاقة الوراثة. ومن ناحية أخرى، قد يكون للنوع الفرعي أنواع فرعية أخرى، وهذا ينتج تسلسلاً هرمياً من الأنواع. قد يضيف النوع الفرعي خصائص جديدة للنوع الأم (مثلاً عمليات

جديدة) و/أو إعادة تعريف عمليات موجودة فيه. إن إضافة عمليات جديدة لا يؤثر في التوابع (Clients) الموجودة أصلاً للنوع لأن هذه التوابع قد تهملها. كما إن إعادة تحديد عملية لن يؤثر في التوابع إذا فرضت لغة البرمجة المُستخدمة بعض القيود كما في لغة جافا (Java). ومن خصائص التركيب الأساسية التي تدعم التطور خاصيتا تعدد الأشكال (Polymorphism) والربط الديناميكي.

تتيح خاصية تعدد الأشكال تحديد متغيرات يمكن أن تشير إلى أنواع عديدة. أي إن الربط بين المتغير ونوع الكائن الذي يشير إليه المتغير يُحدد في فترة التنفيذ. تُحدد هذه المتغيرات (التي تعرف بالمتغيرات متعددة الأشكال) كمرجعية للنوع، وقد تشير إلى كوائن تتبع لأي من الأنواع الفرعية. بكلمات أخرى، يحدد الطراز الثابت من المتغير بواسطة نوع الكائن الذي تشير إليه حالياً. يمكن تحديد الطراز الديناميكي بواسطة أي من الأنواع الفرعية التابعة للنوع الذي يحدد الطراز الثابت. عن طريق الربط الديناميكي، يمكن تحديد العملية التي يتم استدعاؤها بواسطة النوع الذي يحدد طرازها الديناميكي.



الشكل (1 ـ 2): في فترة التنفيذ، لا يمكن التأكد من رقم إصدار الـ Method الذي سيتم استدعاؤه

إن المرونة التي تضمنها عملية الربط قد تسبب بعض المشكلات المتعلقة بالأمن: فبما إن العملية المستدعاة لا يمكن أن تحدد في فترة التجميع، فكيف يمكن ضمان استدعائها بصورة صحيحة في فترة التنفيذ؟ تحتفظ معظم لغات البرمجة كائنية التوجه، بما في ذلك جافا، بمنافع أمن النوع في سياق الربط الديناميكي عن طريق تقييد طريقة إعادة تعريف الـ (Method) في النوع الفرعي. نظراً إلى أن التحقق من الطراز يتم تنفيذه في أثناء فترة التنفيذ من خلال النظر إلى الطراز الثابت للمتغير، فإنه يمكن وجود حل يتمثل في إعادة التعريف

المقيد وليس تغير واجهة ال (Method) الذي يتم إعادة تعريفه[3]. وعليه يمكن أن يكون تركيب البرمجية من نوع التحقق من الطراز بطريقة ثابتة حتى لو كانت الروابط بين الكوائن متغيرة ديناميكياً في فترة التنفيذ.

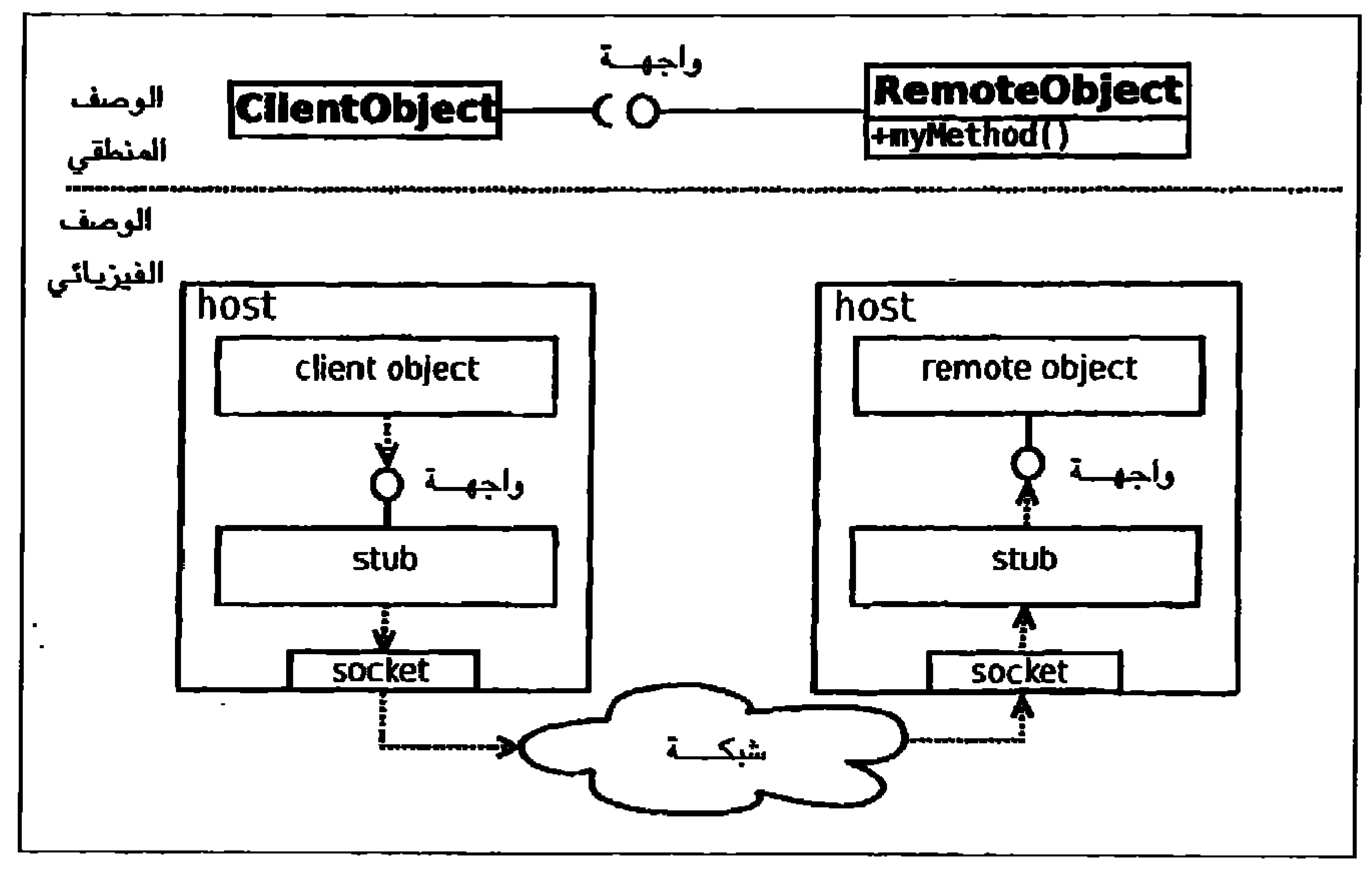
1 ـ 4 ـ 3 المكوّنات والتوزيع

أصبحت المكوّنات الجاهزة للاستخدام[14] متاحة لمطوّري البرمجيات. أدى ذلك إلى طلب إجراء تغييرات في الأساليب المستخدمة لتصميم البرامج: فقد استبدلت منهجيات التصميم التقليدية التي تعتمد على تحليل البرنامج من الأعلى إلى الأسفل جزئياً بمنهجية التكامل من الأسفل إلى الأعلى. وبمجيء المكوّنات الجاهزة للاستخدام أصبح تطوير العمليات غير مركزي، حيث تكون المؤسسات المختلفة مسؤولة عن أجزاء مختلفة من النظام في أوقات مختلفة. إن لذلك مزايا واضحة في عملية التطوير من حيث الكفاءة التي يمكن أن تتقدم بسرعة أعلى، ذلك أن أجزاء كبيرة من الحلول مدعومة بواسطة مكوّنات قابلة لإعادة الاستخدام. تتضمن المسؤوليات غير المركزية أيضاً تحكماً أقل بالنظام كله وذلك بسبب عدم مسؤولية أي مؤسسة منفردة عنه. مثلاً، إن تطور المكوّنات لإدراج خصائص جديدة أو لاستبدال خصائص موجودة أصلاً لا يكون ضمن سيطرة موحّد النظام. تكون المكوّنات في الأغلب جزءاً من نظام موزع على عدة أجهزة. لا شك أن التوزيع خطوة كبرى من مظاهر تطور مكوّنات البرمجيات. بوجود التوزيع، يصبح الربط بين أحد المكوّنات والجهاز الافتراضي الذي ينفّذ ذلك المكوّن قراراً تصميمياً مهماً. تأتي أهمية توزيع البرنامج على عدة أجهزة مرتبطة في الشبكة من الحاجة إلى أن تعكس طبيعة المؤسسة الموزعة في عدة مناطق والتي تستخدم ذلك البرنامج. غالباً ما يتم تجاهل الروابط بين وظائف الأجهزة في أثناء تطوير النظام، لكن يمكن نشرها في أثناء فترة النشر.

في النظام الموزع، ينفذ الربط بين المكوّنات بواسطة وسيط[18]. ومن أشهر الأمثلة على الوسيط (Java RMI)[28] (أي استدعاء المنهجية عن بعد الخاص بـ Java) فهو يدعم هيكليات الخادم ـ التابع بواسطة وسائل من استدعاءات عن بعد (الشكل 1-3). يمكن الوصول إلى كائن (Java) بواسطة أجهزة تعمل عن بعد كما لو كان تنفيذها يتم في الجهاز نفسه. يستدعي الجهاز

(3) يتيح المرجعان 11 و27 تحليلاً متعمقاً لهذه القضايا، ويعطيان معايير عامة لأمن النمط.

التابع ال (Methods) كما لو أن يقوم باستدعائها في كائن محلي. يدرك وسيط (RMI) الرابط عن طريق تنظيم الطلب وإرساله عبر (TCP/IP). ثم يقوم الجهاز الخادم باستقبال الطلب وتنفيذ ال (Method) ويرسل النتيجة بالطريقة نفسها. ومن الأمثلة الأخرى على هيكلية وسيط طلب الكائن المشترك (CORBA)[33] وهذه الهيكلية تدعم المكوّنات الموزعة المكتوبة بعدة لغات برمجة. قد يتم اعتماد سياسيات الربط لدعم التوزيع. ومن الاحتمالات الممكنة تنفيذ ربط ثابت بين أحد المكوّنات وجهاز افتراضي في فترة النشر. وقد يفكر أحدهم في الربط الديناميكي بين أحد المكوّنات وجهاز افتراضي في فترة النشر. وهذه هي الحال في إعادة تهيئة النظام الديناميكي الذي يهدف إلى محاولة تحقيق أفضل تناغم لاعتمادية وأداء البرنامج الموزع عن طريق إعادة ربط المكوّنات بالشبكة في أثناء فترة التنفيذ.



الشكل (1 ـ 3) : التوزيع بين الوصف المنطقي والوصف الفيزيائي مع الوسيط Java RMI

1 ـ 5 تركيب البرمجيات في العالم المفتوح

تتطور عملية تطوير البرمجيات نحو المنهجيات الأكثر مرونة وغير المركزية، حتى تدعم التطور المتطلبات المستمر. لقد تحقق ذلك عن طريق محاولة توقع التغيير مسبقاً ما أمكن، وعن طريق حجب أجزاء كبيرة من التطبيق قيد الاستخدام بحيث لا تتأثر بالتغيير المطلوب تنفيذه في أجزاء أخرى. تعتبر هذه المنهجية مبدأ رئيساً في التصميم ويجب أن يضطلع بها مهندسو

البرمجيات. ومع ذلك، فإن سرعة التطور في وقتنا هذا وصلت إلى مستويات غير مسبوقة، فالحدود بين النظم التي يتوجب تطويرها والعالم الحقيقي الذي تتفاعل معه هذه النظم تتغير باستمرار. في بعض الحالات، لا يمكن توقع التغيير، إذ يحدث التغيير في أثناء تشغيل النظام وعمله، والنظم التي تكون قيد التشغيل يجب أن تكون قادرة على التفاعل مع التغيير بطريقة منطقية. لقد أطلقنا على ذلك «سيناريو العالم المفتوح» مسبقاً.

تحدث سيناريوهات العالم المفتوح في البرامج الحديثة كفهم المحيط (Ambient Intelligence)[17] والحوسبة المتخللة (Pervasive Computing)[40]. في أطر العمل هذه، تكون نقاط التخمين متحركة وتتغير الطوبولوجيا الفيزيائية للنظام بشكل مستمر. كما يتطلب الأمر أن تتغير البنية المنطقية (أي المكوّنات البرمجية) حتى تدعم الخدمات الموقعية. مثلاً، إذا كان شخص يحمل جهاز المساعد الشخصي الرقمي (PDA) أو هاتفاً خلوياً يتجول به في مبنى ويصدر منه أمراً لطباعة وثيقة، فإنه يتم إعادة توجيه الربط إلى مشغّل أقرب طابعة ديناميكياً.

إن الربط المحدد بالموقع ما هو إلا حالة من حالات الربط المحددة بالسياق النظري. أما في حالة فهم المحيط، فقد يتم توفّر معلومات السياق بواسطة أجهزة تحسس (مجسات). مثال على ذلك مجس الضوء الخارجي الذي قد يستشعر وجود ضوء الشمس. قد يتم ربط طلب توفير ضوء أكثر في الغرفة بوحدة برمجية ترسل إشارة إلى مشغل ميكانيكي ليفتح ستائر النوافذ. على نحو معاكس، قد تتسبب العتمة في الخارج بربط طلب التحول إلى أجهزة الإنارة الكهربائية. وقد تأخذ أمثلة أخرى على الربط المحدد بالسياق في الحسبان حالة المستخدم الذي يتفاعل مع المحيط الذي يعمل فيه النظام.

هذا وقد تكون البرامج الجديدة ذات مستويات الأعمال المتقدمة عرضة لتغير المتطلبات باستمرار.

ترتكز نماذج الأعمال الحديثة على فكرة المؤسسات المرتبطة بواسطة شبكات تعمل على توحيد السلوك المتبع ديناميكياً لتحقيق أهداف العمل. قد يتم دعم المؤسسات الموحدة التي تستهدف تحقيق أهدافها بواسطة تركيبات برمجية على مستوى معلومات النظام. في مثل هذه الحال، يتكون تركيب البرمجية الموزع من عناصر تديرها مؤسسات مختلفة. تكشف كل مؤسسة ـ على حِدَة ـ عن جزء من المعلومات الخاص بها مما قد يكون مفيداً لاستخدامات المؤسسات

الأخرى، وفي الوقت ذاته، تستفيد من الخدمات التي تعرضها المؤسسات الأخرى. قد تتغير الروابط بين الخدمات المطلوبة والخدمات المعروضة ديناميكياً.

يتطلب التحول إلى هذه الحالات وجود آليات تركيب جديدة للبرمجيات ومستويات جديدة من استقلالية سلوك النظم، بحيث تكون قادرة على التكيف مع التغيرات عن طريق إعادة تنظيم بنيتها الداخلية. يتضمن ذلك قدرة آليات التركيب على التعافي الذاتي، بشكل أكثر تعميماً، يجب أن تكون مرتكزة على مراقبة الجودة الشاملة للخدمة لتكون في أمثل حال، وقد يتغير ذلك مع الوقت بسبب تغير البيئة المحيطة.

يشرح ما تبقى من هذا القسم بعض الأمثلة على آليات التركيب التي تهدف إلى تحقيق درجة عالية من التقارن والدينامية لتتعامل مع متطلبات العالم المفتوح.

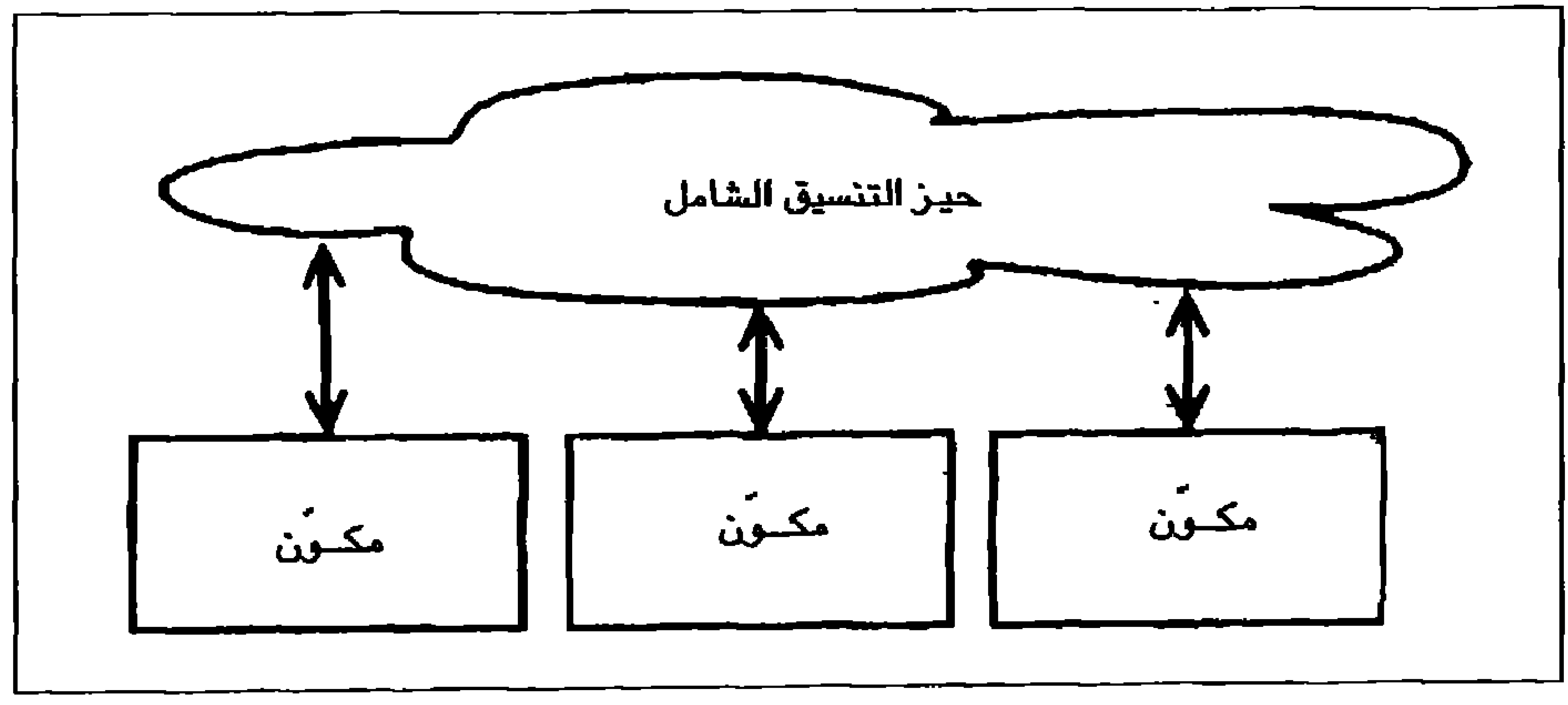
1 ـ 5 ـ 1 حيّز التنسيق الشامل

قد يستخدم وسيط لتوفير حيّز تنسيق شامل (GCS) (الشكل 1-4)، تتفاعل من خلاله المكوّنات مع غيرها، ويتم تنسيق سلوكها بطريقة منفصلة. يتصرف حيّز التنسيق الشامل على أنه وسيط. قد ينتج من كل مكوّن مشترك في الهيكلية بيانات عن تشارك محتمل مع مكوّنات أخرى. هذا ولا تتفاعل المكوّنات مع بعضها البعض بشكل مباشر للتنسيق وتبادل البيانات، بل يتم ذلك من خلال حيّز التنسيق الشامل الوسيط. بناء على ذلك، لا يتوفر لدى المكوّنات المنتجة للبيانات أي معرفة صريحة عن العملاء المستهدفين ولا يكون لها ارتباط مباشر مع غيرها من المكوّنات. يتصرف الوسيط كواسطة بين المكوّنات المنتجة للبيانات ومستخدمي هذه البيانات.

ثمة نوعان من الهيكليات القائمة على حيّز التنسيق الشامل: هيكليات من نوع نشر ـ اشتراك (Publish-Subscribe) وهيكليات من نوع حيّز ـ قائمة مكوّنات (Tuple-Space). يتيح هذان النوعان للنظام أن يكون ديناميكياً حيث إن المكوّنات قد تتحرك جيئة وذهاباً من دون أن تتطلب إعادة تهيئة النظام أو إجراء تغييرات من شأنها التأثير في المكوّنات التي أصبحت جزءاً من الهيكلية. يكون التواصل في هذه الحالة غير متزامن إذ إن المكوّنات المنتجة للبيانات ترسل بياناتها إلى الوسيط وتستمر في تنفيذها من دون انتظار معالجتها بواسطة المستخدم المستهدف.

نظم نشر ـ اشتراك[19] (Publish-Subscribe): تدعم نظم نشر ـ اشتراك

حيّز التنسيق الشامل حيث إن المكوّنات قد تعمل على نشر وقائع معيّنة يتم إرسالها إلى جميع المكوّنات التي أُرسلت معلنة اهتمامها بهذه الوقائع من خلال تقديم اشتراك. يوفّر الوسيط تسهيلات لهذه المكوّنات لتسجيل اهتمامها بوقائع معيّنة من خلال الاشتراك، كما يُرسل الوسيط إشعارات عندما يتم إنشاء الوقائع التي تهتم بها تلك المكوّنات. يعمل حيّز التنسيق الشامل هنا كمحول للوقائع، وهو في هذه الحالة عبارة عن وسيلة عامة منطقياً، بيد أنه قد يتم تطبيقها كبنية تحتية لوسيط موزع بالكامل. ثمة أمثلة عديدة على نظم نشر ـ اشتراك، ذكرت في المواد[25, 16, 14, 13]. ومن هذه النُظم ما هو منتجات صناعية، ومنها ما هو نماذج بحث تتيح استخدام خصائص متقدمة. بعض هذه النظم عبارة عن محول مركزي وبعضها عبارة عن محول موزع. قد تختلف هذه النظم من الناحية الوظيفية، ومن ناحية جودة الخدمة التي تدعمها. فمن الناحية الوظيفية، قد تختلف هذه النظم في أنواع الوقائع التي تُنشئها وفي طريقة تحديد الاشتراكات. مثلاً، من الممكن التمييز بين نظم نشر ـ اشتراك المبنية على المحتوى وتلك التي تكون مبنية على الموضوع. في الحالة السابقة، الواقعة هي عبارة عن كائن ذي خصائص معيّنة يطلب الاشتراك فيها. مثلاً، قد تكون الواقعة عبارة عن إشارة عن توفّر رحلة طيران جديدة بين مدينتي ميلانو وشيكاغو تبدأ من تاريخ معيّن (مثلاً في الأول من كانون الأول) وبأجرة سفر معيّن (مثلاً 400 يورو). قد يحدد الاشتراك حالة اهتمام بمعلومات رحلات الطيران بين مدينتي مدريد ودبلن مثلاً وبأجرة مخفضة (أقل من 70 يورو). في نظم نشر ـ اشتراك المبنية على الموضوع، تكون الوقائع عبارة عن كيانات مفردة. على سبيل المثال، يتم إنشاء الواقعة بواسطة إحدى شركات الطيران (مثلاً الإيطالية للطيران) عندما يعلن عن وجود رحلة طيران معيّنة.



الشكل (1 ـ 4) : حيّز التنسيق الشامل

مثال على ذلك، قد يكون لدينا الوقائع «الإيطالية» أو «المتحدة» أو «الأوروبية». ولا تحمل هذه الوقائع في هذه الحالة أي قيم.

نظم حيّز ـ قائمة مكوّنات (Tuple-Space): تدعم حيّز ـ قائمة مكوّنات (Tuple-Space) التنسيق من خلال مستودع دائم: يوفّر حيّز التنسيق الشامل خصائص تخزين واستعادة البيانات الدائمة. أما أول طرق التنسيق هذه فكان[*] (Linda)[12]. تدعم طريقة ليندا البيانات الدائمة على هيئة قائمة مرتبة من المكوّنات. وقد تكون قوائم المكوّنات المرتبة مكتوبة في حيّز سلسلة المكوّنات؛ كما يمكن قراءتها (بطريقة لا تسبب تدميرها) وحذفها. تحدد عمليات القراءة والحذف قائمة مكوّنات من خلال قالب يستخدم لاختيار قائمة المكوّنات من خلال مطابقة الأنماط. إذا حدد إجراء المطابقة أكثر من قائمة مكوّنات واحدة، يتم اختيار واحدة بطريقة غير حتمية. إذا لم يجد مطابقة، يبقى المكوّن الذي أصدر أمر القراءة أو الحذف في حالة تعليق إلى أن يتم إدخال قائمة مكوّنات مطابقة في حيّز قوائم المكوّنات.

هناك العديد من الفروقات في منهجية ليندا (Linda) الأصلية التي تم تنفيذها من قبل الوسيط المعتمد على نظم حيّز ـ قائمة مكوّنات كـ (Javaspaces)[20]. كما إن هناك تطبيقات مختلفة ملائمة أكثر لمتطلبات النظم الموزعة المتغيرة المكان كـ (Lime)[29].

ثمة أهداف مشابهة لنظم حيّز ـ قائمة مكوّنات ونظم نشر ـ اشتراك، تهدف جميعها إلى توفير دعم مرن لهيكليات البرمجيات الديناميكية. بيد أن نظم نشر ـ اشتراك أقل أهمية: فقد تضاف صفة الديمومة كخدمة لكنها لا تكون مدعومة مباشرة كخاصية أصلية.

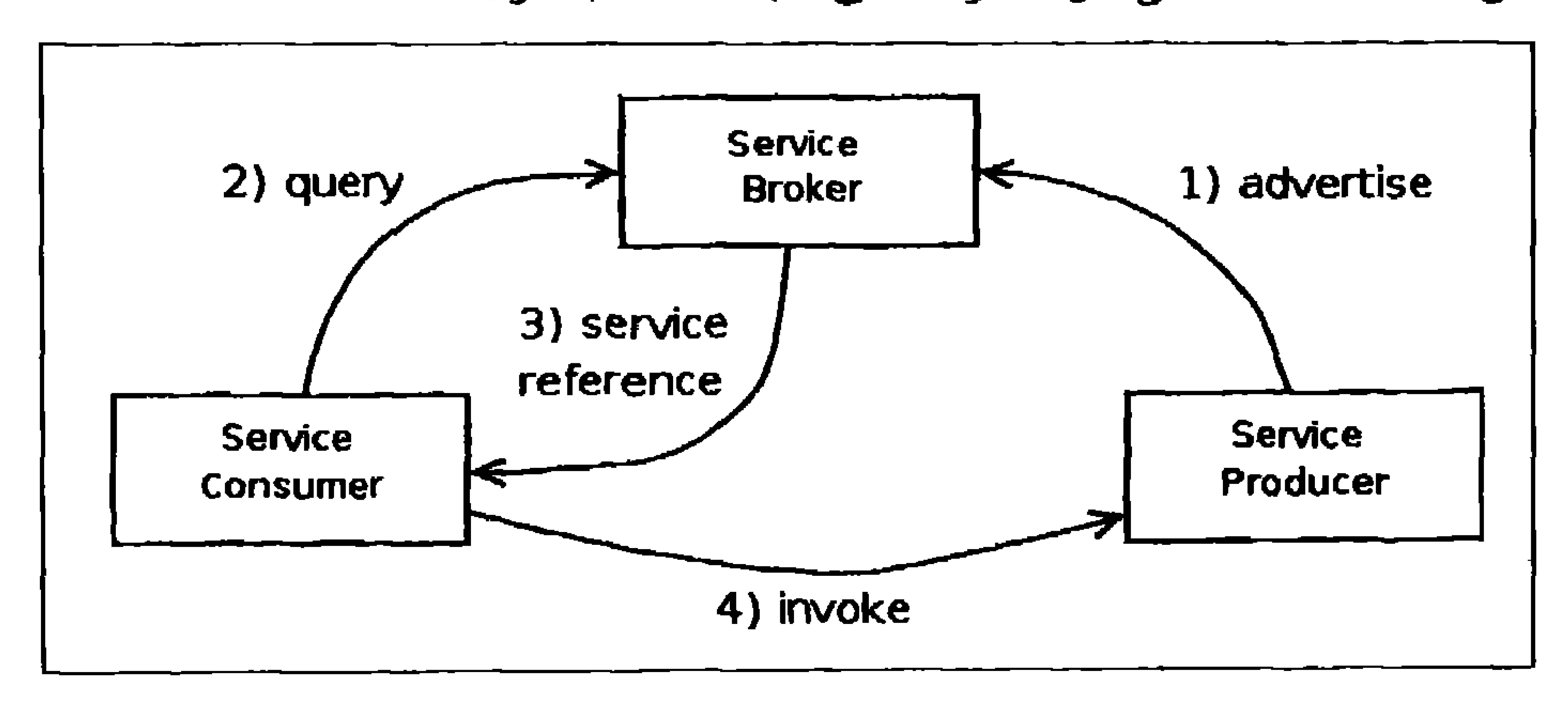
1 ـ 5 ـ 2 الهيكليات خدمية التوجه

يدعم حيّز التنسيق الشامل التقارن والديناميكية عن طريق إلغاء الروابط المباشرة بين مصادر الرسائل ووجهاتها المستهدفة. تحقق الهيكليات خدمية التوجه (SOA)[34] الأهداف نفسها عن طريق توفير تسهيلات تتيح للمكوّنات الإعلان عن توافرها وعن الوظائف التي تؤديها، وتدعم اكتشاف مكوّنات تستطيع تنفيذ الوظائف المطلوبة. حالما يتم اكتشاف هذه المكوّنات، يمكن استخدام

(*) <http://netlib2.cs.utk.edu/pvm3/book/node16.html>.

المكوّن عن بعد من خلال الربط المباشر. من حيث المبدأ، يمكن إجراء عمليات التسجيل والاكتشاف والربط بطريقة ديناميكية في أثناء فترة التنفيذ.

تتكون الهيكليات خدمية التوجه أساسياً من ثلاثة أنواع من المكوّنات (الشكل 1-5): مستهلِك الخدمة، ومزود الخدمة، ووسيط الخدمة. أما مزود الخدمة فهو من يقدم الخدمة. والخدمة توصف من خلال خصائصها (وقد نفترض أنها تتكون من الوصف البيني تقريباً). يعلن المزود عن توفر الخدمة للوسيط الذي يقوم بدوره بعمل رابط للخدمة ويخرنه في الواجهة. أما المستهلِك فهو من يحتاج تلك الخدمة، وهنا يقوم بإرسال طلب إلى الوسيط مرفقاً بمواصفة الخدمة التي يرغب في الحصول عليها. يقوم الوسيط بالاستعلام عن الخدمة، ويحصل مرة أخرى على رابط يصله بالمزود.



الشكل (1 ـ 5): تصميم باتجاه تحقيق خدمة

من هذه النقطة، يستخدم المستهلك العمليات التي يعرضها المزود عن طريق التواصل المباشر معه. توفر الهيكليات خدمية التوجه حلاً جيداً في حالات طلب تفاعلات معقدة بين عناصر مقترنة بعض الشيء والمتطلبات التي قد يُجرى عليها تغيير، وعليه يتطلب الأمر إعادة تنظيم مستمر وسريع للنظام. لقد ذكرنا سابقاً مثالاً على التطبيقات المتنقلة في كل مكان، حيث يمكن تنفيذ مرحلة الاكتشاف ديناميكياً لتحديد التسهيلات المحلية المتاحة لربطها، عند تغير الموقع الفيزيائي في أثناء التنفيذ. كما ذكرنا حالة عن نظم المعلومات المستخدمة في مؤسسات الأعمال المرتبطة بشبكات محوسبة ـ وهي اتحادات الأعمال الديناميكية المتشكلة للاستجابة لفرص الأعمال المتغيرة. في هذه الحالة، تصدّر كل مؤسسة

مشتركة في الاتحاد بعض الوظائف التي تمنح وصولية مسيطر عليها للبرنامج الداخلي. بدوره، قد يستخدم البرنامج الداخلي الخدمات التي تقدمها برامج أخرى والاختيار بين عدة مزودين ممن يوفرون الخدمات نفسها، بناءً على بعض الأفكار الواضحة عن جودة الخدمة المطلوبة والمقدمة. تتوفّر العديد من الحلول الوسيطة التي تطبق الهيكليات خدمية التوجه ـ على سبيل المثال، تقنية شبكة(*) (JINI)[43] أو (OSGI)(**)[3]. على مستوى مؤسسة الأعمال، نشأت خدمات الويب[4]، ومجموعة الحلول القياسية التي ترافقها كطريقة واعدة، ليس في مجال البحث فحسب، بل بين الممارسين أيضاً.

تعرّف (IBM)[43] خدمات الويب على أنها سلالة جديدة من تطبيقات ويب. وهي محتواة في ذاتها وموصوفة بذاتها، كما إنها عبارة عن وحدات تطبيقية يمكن نشرها وتحديد مكان خاص بها واستدعاؤها من خلال الويب. تنفذ خدمات الويب وظائف معيّنة يمكن أن تكون أي شيء تتراوح من الطلبات البسيطة إلى عمليات الأعمال المعقدة... حالما يتم نشر إحدى خدمات الويب، يمكن أن تكتشفها التطبيقات الأخرى (وخدمات ويب أخرى) وتستدعيها... توفّر تقنية خدمات الويب فائدة عظيمة تفوق التقنيات ذات التوجة الخدمي الأخرى ذلك أنها تهدف إلى تسهيل العمليات الواجهة بين البرامج المختلفة.

يفيد ذلك بشكل خاص في البيئات المفتوحة، حيث يكون من المستحيل إجبار جميع المشاركين على تبنّي تكنولوجيا محددة لتطوير أنظمتهم. تتكون تقنية خدمات الويب من مجموعة من المعايير التي تحدد بروتوكولات الاتصال والواجهات التي تستوردها الخدمة من دون فرض قيود على التنفيذ الداخلي للخدمات. أما الأمر العكسي، فإن الحلول الأخرى ـ كـ (JINI)، فتعتمد على لغة البرمجة جافا بهدف التواصل وتنفيذ الخدمات. تتيح التوافقية تكاملاً انسيابياً لحلول خدمات الويب المطوّرة من قبل مؤسسات مختلفة. يعتمد التواصل بين

(*) هي هيكلية خاصة بالشركات لبناء النظم الموزعة على هيئة خدمات خاصة بالوحدات المتشاركة. طورتها في الأساس شركة (Sun)، حيث أصدرتها كإصدار تجاري. تم نقل (JINI) بعد ذلك لشركة (Apache) تحت الاسم (River). توفر (JINI) تسهيلات للتعامل مع بعض الأفكار الخاطئة حول نظم الحوسبة الموزعة ومشكلات تطور النظم والمرونة وأمن النظم ومكوناتها (المترجم).

(**) إطار العمل (OSGI) هو نظام وحدة وبرنامج خدمي يستخدم في لغة البرمجة (Java) الذي ينفذ وحدة مكونات ديناميكية وكاملة. يمكن تركيب التطبيقات والمكونات عن بعد ومن ثم بدئه وتحديثه وإعادة تركيبه من دون الحاجة إلى إعادة تشغيل الجهاز (المترجم).

خدمات الويب على بروتوكول وصولية الكائن البسيطة [SOAP][10] الذي يصف بناء الرسالة النصية التي يمكن تبادلها على هيئة استدعاءات للوظيفة. يعتمد هذا البروتوكول على لغة الترميز القابلة للامتداد (XML)، ويتم تنفيذه من خلال بروتوكول نقل النصوص التشعبية (http)، ما يحد من التدخل في نظم المعلومات لتوريد الخدمات. بعد اكتشاف توفر الخدمة وحالما يتم الربط بين مستهلك الخدمة ومزودها، يتحقق التواصل عن طريق إرسال رسالة من خلال بروتوكول وصولية الكائن البسيطة. يمكن رؤية الخدمات على أنها تطور في المكوّنات الجاهزة للاستعمال. أما وظائف تطبيق الاستعمالات الممكنة وقيمتها بالنسبة إلى الآخرين فتتجه نحو التطوير اللامركزي، ذلك أن المؤسسة المسؤولة عن تطوير المكوّنات الجاهزة للاستخدام تختلف عن المؤسسات التي تستخدمها عموماً. في كلتا الحالتين، ثمة منافع واضحة لذلك (تقليل فترة التطوير وزيادة الاعتمادية)، لكن هناك أيضاً مساوئ ممكنة لعمليات التطوير التي تتم بأكملها داخل المؤسسة نفسها (اعتمادها على أطراف أخرى لإحداث التطوير مستقبلاً). الفرق الوحيد بين المكوّنات الخدمية والمكوّنات الجاهزة للاستخدام هو أن الأخيرة تكون جزءاً من البرنامج ويتم تنفيذها في مجال ذلك البرنامج. أما الخدمات فهي ملك لمزود معيّن وهو مسؤول عن تنفيذها. تمثل آليات التركيب وجهاً آخر من أوجه الاختلاف، ففي حالة الخدمات، يمكن اختيار الأهداف الممكنة للربط وتنفيذ هذه الروابط ديناميكياً في أثناء فترة التنفيذ.

أما الاختلاف الأكثر لفتاً فهو درجة التحكم والثقة. فالمكوّنات الجاهزة للاستخدام لا يمكن تغييرها بعد أن تصبح جزءاً من البرنامج إلى أن يقوم مالك البرنامج باستبدالها. من ناحية أخرى، تكون الخدمات ملكاً للمزودين وهم من يتحكم بتطورها. وبناءً على ذلك يمكن تغيير الخدمات لمنفعة المستخدمين.

على الرغم من أن الخدمات أصبحت منهجاً عملياً في هيكليات النظم القابلة للتطور، لا يزال هناك العديد من المشكلات التي يجب حلها قبل تطبيق هذه الخدمات في بيئات مفتوحة أو في الحالات التي يجب أن تلبّي فيها النظم متطلبات الاعتمادية الصارمة.

يبيّن القسم الآتي أهم التحديات التي يجب أن تحققها الهيكليات التي تدعم تراكيب البرمجيات الديناميكية بشكل عام، وعلى وجه الخصوص من قبل الهيكليات خدمية التوجه.

1 ـ 6 التحديات والعمل المستقبلي

ناقشنا عدداً من المنهجيات المتبعة حالياً والمنهجيات الواعدة التي تدعم تركيب البرمجيات في العالم المفتوح. كما لاحظنا، عندما تصبح الحلول ديناميكية أكثر وغير مركزية، حُلّ العديد من المشكلات للوصول إلى الدرجة الضرورية من الاعتماد المطلوب عملياً. نحاول في هذا القسم تحديد أين تكمن المشكلات الرئيسة واتجاهات البحث الممكنة. بعض هذه المشكلات جديد، والعديد منها كان وما زال موجوداً لكنها تصبح أكثر صعوبة في الإعدادات الجديدة.

التحدي الأول: فهم العالم الخارجي. كانت المتطلبات وما زالت مشكلة حاسمة بالنسبة إلى مهندسي البرمجيات[42, 32]. إن اختيار وتحديد المتطلبات في ظل الظروف دائمة التغيير أصبح أكثر صعوبة. في حالة النظم المتفشية، يتطلب الأمر أن تكون الحلول ذات قابلية تكيّف عالية. مثلاً، يتطلب دعم كبار السن أو المعوقين جسدياً في بيوتهم تطوير حلول يمكن تكييفها وتخصيصها لاحتياجات معيّنة تتواءم مع مستخدميها. على مستوى الأعمال، يجب التقاط متطلبات الشركات الفيدرالية بطريقة ملائمة لدعم عملياتها الواجهة. من الأهمية بمكان تحديد ما يجب أن يتم عرضه للآخرين كخدمات وما يجب الاحتفاظ به وحمايته كملكية خاصة. إن التكامل السهل والديناميكي ضروري لالتقاط فرص الأعمال بسرعة. بالرغم من أن ذلك موضوع للبحث في المؤسسات التجارية، إلا أنه يتوجب على مهندسي البرمجيات إدراك هذه التوجهات، إذ يجب أن يكونوا قادرين على التقاط المتطلبات الخاصة بالشركات الديناميكية وتشكيل الحلول الهيكلية بناء على ذلك.

التحدي الثاني: دعم عمليات البرمجيات. تتسارع مؤسسات الأعمال باطّراد في طريقة استجابتها لفرص الأعمال. هذا ويجب أن تتقدم عملية تطوير البرمجيات بالنهج السريع نفسه لتوفير الحلول بسرعة وذلك لتحقيق المزايا التنافسية وتقليل الوقت اللازم للتسويق. أما كيفية استجابة أساليب التطوير السريعة لمعايير الاعتمادية العالية التي قد تكون متطلباً للبرامج فلا تزال قضية مفتوحة. تصبح المشكلات أصعب لأن فرق التطوير أصبحت موزعة ومستقلة[24, 9, 1]. كيف يمكن تفعيل الممارسات المعيارية؟ ما هي محركات الإنتاجية الأساسية؟ كيف يمكن تعريف جودة العملية وكيف يمكن قياسها؟

التحدي الثالث: مواصفات الخدمة. إن التطور المستمر نحو الخدمات

التي يمكن تطويرها ثم نشرها لتستخدمها جهات أخرى تجعل من المهم توفير مواصفات لتلك الخدمة بحيث يستطيع المستخدمون فهمها. كانت مواصفات البرمجيات وما زالت موضع بحث نشط. يعود تاريخ الحاجة إلى مواصفات دقيقة يمكن أن تستخدمها برامج العملاء[35]. المشكلة حاسمة في حالة الخدمات بسبب الفصل التام بين الموردين والمستخدمين وبسبب الحاجة إلى أن تكون المواصفات موثقة بطريقة أقوى وأكثر قابلية للتنفيذ. إذا كان الأمر متعاقداً عليه، يجب أن توفر المواصفات بياناً دقيقاً أكثر من مجرد تعريف صرف ونحوي لواجهة الاستخدام وأكثر تحديداً للوظائف. كما يجب أن ينص على الجودة التي تضمنها الخدمة.

يجب أن تكون المواصفات قابلة للبحث. كما يجب أن يتم تحديدها من حيث التجميعات المشتركة[2]. أخيراً، يجب أن تكون قابلة للتأليف؛ بمعنى أنها يجب أن يكون اشتقاق مواصفات إنشاء الخدمة التي تعرف مواصفاتها أمراً ممكناً.

التحدي الرابع: البرمجة والتركيب. لقد قمنا بوصف النماذج المختلفة لتركيب الخدمة كنظم نشر ـ اشتراك ونظم حيّز ـ قائمة المكوّنات والهيكليات خدمية التوجه. يتراوح التنسيق بين المكوّنات من التراكيب المتسقة حيث يعمل مخطط سير العمل على تنسيق تنفيذ الخدمة حالما يتم اكتشافها وربطها، إلى التراكيب المصممة كنمطي نشر ـ اشتراك وحيّز ـ قائمة مكوّنات، حيث يجب أن تتواءم الخدمات مع البروتوكول المشترك للرسائل المتبادلة والمضي قدماً بطريقة نظير ـ نظير[30, 26]. كيف يمكن لآليات التركيب المختلفة أن تتكامل وتندمج مع لغة البرمجة؟ كيف يمكن للغة البرمجة دعم مدى كامل من آليات الربط المرنة من فترة ما قبل التنفيذ الثابتة إلى الديناميكية الكاملة؟ كيف يمكن أن تتضمن سياسات الربط عمليات التفاوض واستراتيجيات إعادة الربط في حال وجود تباين عن جودة الخدمة المستهدفة المعلن عنها؟ كيف يمكن أن تنشأ سلوكيات الالتئام الذاتي أو التنظيم الذاتي من خلال التراكيب الملائمة؟

التحدي الخامس: الثقة والتحقق. إلى هنا تكون الخدمات قد طُورت وتم تشغيلها في نطاقاتها الخاصة بها. لا يملك مستخدمو الخدمة أي سيطرة عليها، كما إنهم لا يملكون صلاحية الوصول إلى باطنها. إذا كانت الخدمات مرتبطة في فترتي الترجمة والنشر فقط، ولا يحدث عليها أي تغيير بعد ذلك، فإنه يمكن أن تخضع التطبيقات لإجراءات التحقق التقليدية. لكن في حالة الربط الديناميكي، ونظراً إلى

أن تنفيذ الخدمة قد يتغير بطريقة غير مصرح بها، فإن المنهجيات التقليدية تفشل.

يجب أن تتم عملية التحقق في أثناء فترة التنفيذ. كما يجب أن تضمن عمليات مراقبة الخدمة المستمرة[6] إذا ما تم الكشف عن أي تباين عن جودة الخدمة المتوقعة وأن الإجراءات الملائمة قد اتخذت. بشكل عام، يجب أن يكون المُستخدم قادراً على إيجاد طرق لضمان الاعتماد على الخدمة وجعلها آمنة كما تتطلبه المتطلبات. إن بناء نظم آمنة وقابلة للاعتماد عليها من خدمات ذات موثوقية منخفضة هو تحدٍ كبير.

التحدي السادس: الاتساق. في البيئات المغلقة، يكون التطبيق الصحيح في حالة متسقة دائماً مع البيئة التي يتواجد فيها. أما في البيئات المفتوحة، تتغير البيئة مسببةً تغيراً في البرمجيات وينتج من ذلك حالة عدم اتساق. هذا وقد أدرك العديد من الباحثين الحاجة إلى إدارة عدم الاتساق كإشكالية في السابق[2, 15, 5]. من الأهمية بمكان التعامل معها في حالة النظم المتطورة ديناميكياً.

كانت تلك قائمة قصيرة تضمنت أهم التحديات التي يجب أن تكون جزءاً من أجندة البحث في مجالات هندسة البرمجيات. هذا ويجب أن لا يُنظر إلى هذه القائمة على أنها مضنية، إذ إنها تركز فقط على بعض المشكلات التي تعمل عليها مجموعتنا. يجب أن يستمر التطور في هذه المجالات وغيرها لتحسين جودة التطبيقات البرمجية في مجالات منبثقة جديدة.

شكر وتقدير

لقد تأثر هذا العمل بالخبرة المكتسبة من مشروع (ART DECO) الوطني ومشروع هندسة النظم ذات الخدمات المركزية (SeCSE) المموّل من الاتحاد الأوروبي. لقد ساعدت إليزابيتا دي نيتو ولوتشيانو باريسي في تطوير رؤيتنا لتركيب البرمجيات.

المراجـع

1. Flexible and distributed software processes: Old petunias in new bowls?

2. Owl specification. <http://www.w3.org/TR/owl-ref>.

3. OSGI Alliance. *OSGI Service Platform: Release 3, March 2003*. IOS Press, Amsterdam, 2003.

4. G. Alonso [et al.]. *Web Services: Concepts, Architectures and Applications.* Berlin: Springer, 2004.

5. R. Balzer. «Tolerating Inconsistency.» paper presented at: Proceedings of the 13th International Conference on Software Engineering, pp. 158-165, 1991.

6. L. Baresi, C. Ghezzi, and S. Guinea. «Smart monitors for composed services.» paper presented at: Proceedings of the 2nd International Conference on Service Oriented Computing, pages 193-202, 2004.

7. L. Baresi, E. Di Nitto, and C. Ghezzi. «Toward Open-World Software: Issue and Challenges.» *IEEE Computer*: vol. 39, no. 10. 2006, pp. 36-43.

8. F. L. Bauer, L. Bolliet, and H. J. Helms. «Report on a Conference Sponsored by the NATO Science Committee.» In NATO Software Engineering Conference 1968, p. 8.

9. B. W. Boehm and R. Turner. *Balancing Agility and Discipline: A Guide for the Perplexed.* Addison-Wesley, Reading, MA, 2004.

10. D. Box [et al.]. Simple Object Access Protocol (SOAP) 1.1.

11. L. Cardelli and P.Wegner. «On understanding types, data abstraction, and polymorphism.» *ACM* Computing Surveys (CSUR): vol. 17, no. 4, 1985, pp. 471-523.

12. N. Carriero and D. Gelernter. Linda in Context. *Communications of the ACM*: vol. 32, no. 4, 1989, pp. 444-458.

13. A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. «Achieving scalability and expressiveness in an Internet-scale event notification service.» paper presented at: *Proceedings of the Nineteenth Annual ACM* Symposium on Principles of Distributed Computing, 2000, pp. 219-227.

14. G. Cugola, E. Di Nitto, and A. Fuggetta. «The JEDI event-based infrastructure and its application to the development of the OPSS WFMS.» *IEEE Transactions on Software* Engineering: vol. 27, no. 9, 2001, pp. 827-850.

15. G. Cugola [et al.]. «A framework for formalizing inconsistencies and deviations in human-centered systems.» *ACM Transactions on Software* Engineering and Methodology: vol.5, no.3, 1996, pp. 191-230.

16. G. Cugola and G. P. Picco. *REDS: A Reconfigurable Dispatching System: Technical report*, Politecnico di Milano, 2005.

17. K. Ducatel [et al.]. Scenarios for ambient intelligence in 2010 (ISTAG 2001 Final Report). IPTS, Seville, 2000.

18. W. Emmerich. *Engineering Distributed Objects*. New York. John Wiley & Sons, 2000.

19. P. T. H. Eugster [et al.]. «The many faces of publish/subscribe.» *ACM Computing Surveys*: vol. 35, no. 2, 2003, pp. 114-131.

20. E. Freeman, K. Arnold, and S. Hupfer. *JavaSpaces Principles, Patterns, and Practice*. Essex, UK: Addison-Wesley Longman Ltd., 1999.

21. C. Ghezzi and M. Jazayeri. *Programming Language Concepts*. New York: John Wiley & Sons, 1997.

22. C. Ghezzi, M. Jazayeri, and D. Mandrioli. *Fundamentals of Software Engineering*. Englewood Cliffs, NJ: Prentice-Hall, 2003.

23. C. Ghezzi and B. A. Nuseibeh. «Special Issue on Managing Inconsistency in Software Development.» *IEEE Transactions on Software Engineering*: vol. 24, no. 11, 1998, pp. 906-907.

24. J. D. Herbsleb and D. Moitra. Global software development. Software, IEEE 18 (2):16-20, 2001.

25. Tibco Inc. TIB/Rendezvous White Paper, 1999.

26. N. Kavantzas, D. Burdett, and G. Ritzinger. WSCDL: Web Service Choreography Description Language, 2004.

27. B. H. Liskov and J. M. Wing. «A Behavioral Notion of Subtyping.» *ACM Transactions on* Programming Languages and Systems (TOPLAS): vol. 16, no. 6, 1994, pp. 1811-1841.

28. Sun Microsystems. Java Remote Method Invocation Specification, 2002.

29. A. L. Murphy, G. P. Picco, and G. C. Roman. «Lime: A Middleware for Physical and Logical Mobility.» paper presented at: *Proceedings of the 21st International Conference on Distributed Computing* Systems, 2001, pp. 524-533.

30. N. Busi [et al.]. «Choreography and orchestration conformance for system design.» Paper presented at: *Proceedings of 8th International Conference on* Coordination Models and Languages (COORDINATION06) LCNS 4038, pp. 63-81, 2006.

31. P. Naur, B. Randell, and J. N. Buxton. «Software Engineering: Concepts and Techniques.» paper presented at: *Proceedings of the NATO Conferences*. Petrocelli/Charter, New York, 1976.

32. B. Nuseibeh and S. Easterbrook. «Requirements engineering: A roadmap.» paper presented at: *Proceedings of the Conference on The Future of Software Engineering*, 2000, pp. 35-46.

33. R. Orfali and D. Harkey. *Client/Server Programming with Java and CORBA*. New York: John Wiley & Sons, 1998.

34. M. Papazoglou and D. Georgakopoulos. «Service-oriented computing: Introduction.» *Communications of ACM*: vol. 46, no. 10, 2003, pp. 24-28.

35. D. L. Parnas. «On the Criteria to be Used in Decomposing Systems into Modules.» *Communications of the ACM*: vol. 15, no. 12, 1972, pp. 1053-1058.

36. D. L. Parnas. «On the Design and Development of Program Families.» *IEEE Transactions on Software Engineering*: vol. 2, no. 1, 1976, pp. 1-9.

37. D. L. Parnas. «Designing software for ease of extension and contraction.» paper presented at: *Proceedings of the 3rd International Conference on Software Engineering*, 1978, pp. 264-277.

38. W. W. Royce. «Managing the Development of Large Software Systems.» paper presented at: *Proceedings of IEEE WESCON*, 1970, pp. 1-9.

39. B. G. Ryder, M. L. Soffa, and M. Burnett. «The Impact of Software Engineering Research on Modern Programming Languages.» *ACM Transactions on Software Engineering and Methodology (TOSEM)*: vol. 14, no. 4, 2005, pp. 431-477.

40. M. Satyanarayanan. «Pervasive computing: Vision and challenges.» *Personal Communications, IEEE* [see also *IEEE Wireless Communications*]: vol. 8, no. 4, 2001, pp. 10-17.

41. C. Szyperski. *Component Oriented Programming*. Berlin: Springer, 1998.

42. A. Van Lamsweerde. «Requirements Engineering in the Year 00: A Research Perspective.» paper presented at: *Proceedings of the 22nd International Conference on Software Engineering*, 2000, pp. 5-19.

43. J. Waldo. «Jini architecture for network-centric computing.» *Communications of the ACM*: vol. 42, no. 7, 1999, pp. 76-82.

2

التركيب في خطوط إنتاج البرمجيات

كريستيان بريهوفير (Christian Prehofer)
جيليس فان غيرب (Jilles Van Gurp)
جان بوش (Jan Bosch)

2 ـ 1 مقدِّمة

لقد حظيت خطوط إنتاج البرمجيات أو البرامج مؤخراً ببعض الاهتمام في مجالَي البحث والصناعة. انتقل العديد من الشركات من تطوير كل منتج من المنتجات البرمجية من الصفر إلى التركيز على القواسم المشتركة بين المنتجات المختلفة ووضع ذلك ضمن هيكلية المنتج وفي مجموعة مرتبطة من الموجودات التي يمكن إعادة استخدامها. عملية التطوير هذه هي عملية منطقية ذلك أن البرمجيات أصحبت تشكل جزءاً كبيراً من المنتجات وهي ما يحدد الخصائص المنافسة للمنتج في الأغلب؛ خصوصاً في صناعة النظم المضمنة. عند الانتقال من الأجزاء الهامشية إلى الأجزاء الكبرى في المنتج، يصبح الجهد الذي يجب بذله في تطوير البرمجيات أمراً مهماً أيضاً، إذ تبحث الصناعات عن طريق لزيادة إعادة استخدام البرمجيات المتواجدة أصلاً للحد من تطوير برمجيات الخاصة وزيادة جودة البرمجية. لقد أبدى عدد من المؤلفين خبرة في هيكليات خطوط الإنتاج. يعود تاريخ العمل في ذلك إلى أواخر عقد التسعينيات من القرن الماضي[12, 7, 2]، ثم تطور هذا القطاع من ذلك الحين باتجاهات جديدة لزيادة تطبيقات المفهوم.

يلقي هذا الفصل الضوء على عدة تحديات نتجت من زيادة نطاق خطوط الإنتاج الناجحة وتطور مفاهيم جديدة لمنهجية تركيبية لخطوط إنتاج

البرمجيات(14, 13, 5). إضافة إلى ذلك، نوقشت الآثار المترتبة على تطبيق هذه المنهجية على جوانب مختلفة من تطوير البرمجيات، بما في ذلك التساؤلات الإدارية وتلك المتعلقة بالعمليات.

بالرغم من أن مفهوم منصات البرمجيات(*) (Software Platforms) سهل الفهم نظرياً، إلا أنها تواجه تحديات مهمة عملياً. كما ناقشنا في المرجع(3)، يفترض أن يلتقط نموذج المنصة الوظائف الأكثر عمومية والأقل تميزاً. مع ذلك، لا تأخذ الوظائف المخصصة بالمنتج في الاعتبار الحدود بين المنصات والبرمجية التي تعمل ضمن تلك المنصة. إن التطور في النظم المضمنة قد ينتج من الآلات أو المعدات أو البرمجيات. أما التطور في المعدات والآلات فيؤثر في حزم البرمجيات. نظراً إلى أن واجهة الوصول للمعدات تكون ضمن برامج تشغيل الأجهزة في نهاية الحزمة بينما تكون البرامج المتأثرة وواجهات الاستخدام في أعلى الحزمة، يكون للتغيرات التي تطرأ على الآلات والمعدات تأثير شامل يسبّب تغيراً في عدة مواقع أعلى أو أسفل حدود المنصة.

ومن المصادر الأخرى التي تسبب تأثيراً شاملاً البرمجيات الخاصة. فالمنتجات الجديدة تتيح تحديد سيناريوهات استخدام جديدة تعرّف احتياجات جديدة للبرمجيات التي لا يمكن حصرها في عنصر واحد أو تطبيق واحد، بل لها تأثير في المستوى الهيكلي. من الأمثلة على ذلك إضافة خصائص الأمن وإضافة أطر عمل لواجهة استخدام أكثر تطوراً أو أطر العمل ذات الطبيعة الخدمية من خلال الويب.

لقد نجحت عائلات المنتجات البرمجية في حالات عديدة في الشركات التي طبقت استخدامها. نظراً إلى ذلك النجاح، بدأت الشركات تطوير تلك البرمجيات لإتاحة استخدامها في أمور تتعدى نطاقها الأوّلي. يعزى سبب ذلك إما لأن الشركة ترغب في تحديد نطاق المنتجات حسب مدى تقاربها أو بسبب أن نجاح عائلات المنتجات المسبق قد أدى إلى وجود منتجات غير ذات علاقة ببعضها البعض، ضمن عائلة واحدة. يسبّب ذلك وضعاً تصبح فيه عائلة المنتج البرمجي ضحية نجاحها الشخصي. فعندما يتسع نطاق المتطلبات وتتنوع المنتجات المدعومة،

(*) المنصّة في الحوسبة، هي البنية التحتية التي تسمح لبرنامج أو عتاد لأن يعمل. أهمها: البنية المعمارية للحاسوب ونظام التشغيل أو لغات البرمجة والمكتبات التشغيلية. إن كل حاسوب يتكوّن من معالج ونظام تشغيل اللذين يشكّلان منصّة التشغيل (المترجم).

تتسبب المنهجية ذات التوجة التكاملي الأصلية في العديد من المشكلات الخطيرة سواء كان ذلك في النواحي التقنية أو العمليات أو التنظيمية أو الأعمال.

إن المنهجية، ذات التوجه التكاملي، التقليدية المتبعة في خطوط إنتاج البرمجيات تخصص البحث والتطوير إلى مؤسسة تقوم بتسليم البرنامج كنظام برمجي متكامل ومُختبَر مرفقاً به جميع API(*) التي يمكن استخدامها من قبل فرق العمل لاشتقاق المنتجات منها. إضافة إلى ذلك، ثمة سيطرة هيكلية من الأعلى إلى الأسفل تقوم بها مؤسسة المنصة المركزية، بما في ذلك الإدارة المركزية للمتطلبات وخارطة طريق وتحكم تام بالمزايا وحالات التغيير واشتقاقات المنتج، إضافة إلى دمج وتكامل واختبار المنتج ومشتقاته.

تعتبر منهجية المنصة الهيكلية شكلاً محسناً لمنهجية المنصة ذات التوجة التكاملي[5]، حيث تستخدم المنصة ويتم إضافة الوظائف الإضافية بحيث لا تعدل الوظائف الأساسية. يكون ذلك ملائماً فقط عندما تكون الوظائف الأساسية محددة جيداً وكافية لإنشاء المنتج بسرعة. ثانياً، ذلك لا يحد من مشاركة شيفرة البرمجة بين منتجات مختلفة ومشتقة خارج إطار البرنامج الأساسي.

على الرغم من نجاح منهجية المنصة ذات التوجه التكاملي، إلا أننا نجادل في هذا الفصل في ما إذا كانت المنهجية تعاني بعض المحددات عندما يبدأ مجال المنتج بالتزايد. ومن ردود الأفعال الحدسية التي تبنتها الشركات اعتماد منهجية المنصة الهيكلية. على كلٍّ، وكما ناقشنا في المرجع[5]، تفشل هذه المنهجية عند وجود مدى واسع من المنتجات وتباين غير متوقع وفي حالة اشتقاقات المنتج، التي تتطلب تعديل أجزاء محددة في النظام وإعادة تصميمها.

من التساؤلات الأساسية التي تُطرح عن منهجية المنصة التقليدية هو أننا رأينا المؤسسات التي تعتمد المنصة ووحدات المنتج تواجه بعض العقبات في ما يخص التكامل. **أولاً**، نظراً إلى أنه يتم تنفيذ التكامل والاختبار في المنصة

(*) واجهة برمجة التطبيقات (Application Programming Interface) واختصارها (API): هي مجموعة من الروتينات، وهياكل البيانات و/ أو البروتوكولات التي تقدمها المكتبات و/ أو نظام تشغيل الخدمات لدعم بناء البرامج. هناك نوعان منها: يعتمد أحدهما على لغة البرمجة؛ إنه متاح فقط في لغة برمجة معيّنة، ويقوم على استخدام (Syntax) وعناصر هذه اللغة لجعله ملائماً للاستخدام في هذا السياق. والآخر مستقل عن اللغة وهذا يعني أنه مكتوب بطريقة تتيح استخدامه في العديد من لغات البرمجة. وهذا النمط مطلوب في أنواع الواجهات البرمجية (API) المستخدمة في الخدمات غير المرتبطة بعملية معيّنة أو نظام تشغيل، وعادة ما يكون متاحاً كروتين منفصل. مثال عن النوع الثاني الموقع الذي يعرض أماكن تواجد المطاعم في مكان ما (المترجم).

عند مستوى المنتج، فقد يقود ذلك إلى تكلفة مرتفعة جداً لتنفيذ التكامل إذا كان نطاق خط إنتاج المنتج يتسع. إن الاعتماد الكبير على التكامل المتكرر يؤدي أيضاً إلى عدم القدرة على التوقع وعدم الكفاءة. **ثانياً**، يعرقل النطاق الواسع من وظائف المنصة مرونة إصدار مزايا جديدة إلى السوق ويزيد من الوقت اللازم لذلك. **ثالثاً**، يؤدي ذلك إلى فقدان المرونة وإلى دورات تطوير طويلة غير مقبولة، بسبب هشاشة المنصة الناتجة من عدم قدرتها على تجاوز نطاق المتطلبات الأساسي. **أخيراً**، لا يمكن لبرمجية تم تطويرها ضمن منتج معيّن أن يعاد استخدامها بطريقة ملائمة من قبل وحدات أخرى قبل أن يتم تضمنها في إصدار لوحدة المنصة. ذلك أن وحدة المنصة لها دورة إصدار طويلة عادة بحيث يصعب إعادة استخدام البرمجية في مثل هذه المنهجية. بعبارة أخرى، علينا أن نتيح إمكانية المشاركة الأفقية بين المنتجات إضافة إلى المشاركة العمودية بين المنصة والمنتجات.

ومن الحقائق المهمة الأخرى الأهمية المتزايدة للمكوّنات البرمجية ذات المصادر المتاحة للاستخدام والمكوّنات الجاهزة للاستخدام. أدى ذلك إلى نشوء عدد من التحديات التي تواجه المنهجيات التقليدية لأن هذه المكوّنات الجاهزة للاستخدام تتطلب منهجيات أكثر انفتاحاً لا تفترض تحكماً كاملاً بالمزايا والخطط والأدوات.

إن تطوير البرمجيات عامل منافسة وتمييز أساسي في العديد من المجالات كالأجهزة التي تعتبر جزءاً من البرمجية، إذ إن هناك حاجة شديدة لمنهجية جديدة. نجادل هنا أننا نحتاج إلى نقلة نوعية جذرية في طريقة تصميم المنتجات البرمجية إذا كنا سنحقق تحسيناً مهماً في الإنتاجية ووقت الإصدار للسوق.

يعرض هذه الفصل منهجية أكثر مرونة وأكثر انفتاحاً بالتفصيل وهي تعرف بمنهجية عائلة المنتج التركيبية، التي تتناول المسائل المذكورة سابقاً. الفكرة الأساسية للمنهجية التركيبية لخطوط إنتاج البرمجيات هي أن المنصة البرمجية لخط الإنتاج ليس بالحلول البرمجية المتكاملة بشكل كامل، بل هي عبارة عن مجموعة من المكوّنات والإرشادات الهيكلية والمبادئ، إضافة إلى دعم الاختبار والتوثيق وسيناريوهات الاستخدام. بناء على بيئة المنصة المفتوحة والمرنة، يتم تكوين المنتجات بأكملها ودمجها واختبارها. حيث إن تكنولوجيا المكوّنات البرمجية تؤدي دوراً أكبر في هذه المنهجية، نقترح في هذا الفصل استخدام ما

يعرف بـ «مقاطع الهيكلية»، وهي عبارة عن مقاطع من الهيكلية الكاملة. هذه المقاطع الهيكلية تغطي أحد النظم الفرعية أو أكثر من الهيكلية. بهذه الطريقة يمكننا ضمان أن يكون الاختبار والتكامل أكثر من مجرد اختبار على مستوى المكوّنات. وهذا مهم أيضاً في ما يتعلق بالمتطلبات غير الوظيفية خارج نطاق المكوّنات المفردة. هذه العملية موضحة في الشكل 2 ـ 1 من الأمور الحاسمة هنا أن المقاطع الهيكلية لا تكون متكاملة كلياً، كما يجب أن تكون المكوّنات الخارجية التابعة واضحة تماماً.

إن تبنّي منهجية عائلة المنتج التركيبية يسبب تحولاً متعمداً في مسؤوليات دمج واختبار المنتجات، حيث تتكون هذه المنتجات من المكوّنات آنفة الذكر. في تجربتنا هذه، تلائم هذه المنهجية عائلات المنتج ذات النطاق الواسع والتطور السريع والابتكار المفتوح والمنافسة على مستوى المكوّنات.

لقد تمّ تقديم الحافز والخصائص الأساسية لمنهجية عائلات المنتج التركيبية في المرجع الخامس. تم تفصيل المنهجية وتحسينها في هذا الفصل، تحديداً مفهوم مقطع الهيكلية. أما ما يسهم به هذا الفصل فهو ما يأتي: أولاً، نعرض تقييماً تفصيلياً ودقيقاً للمشكلات المتعلقة بالمنهجية ذات التوجه التكاملي. ثانياً، نتعلم نتائج تبنّي منهج عائلة المنتج التركيبية على كافة النواحي المتعلقة بتطوير البرمجيات بما في ذلك المتطلبات والعمليات والتنظيم والهيكلية والأدوات.

أما بقية هذا الفصل فهي منظمة على النحو التالي: في القسم التالي، قدمنا المنهجية ذات التوجه التكاملي وناقشنا سلبياتها وعرضنا المنهج التركيبي كبديل. بعد ذلك، في القسم 2 ـ 3، تم عرض مفهوم مقاطع الهيكلية كعنصر أساسي للمنهج التركيبي.

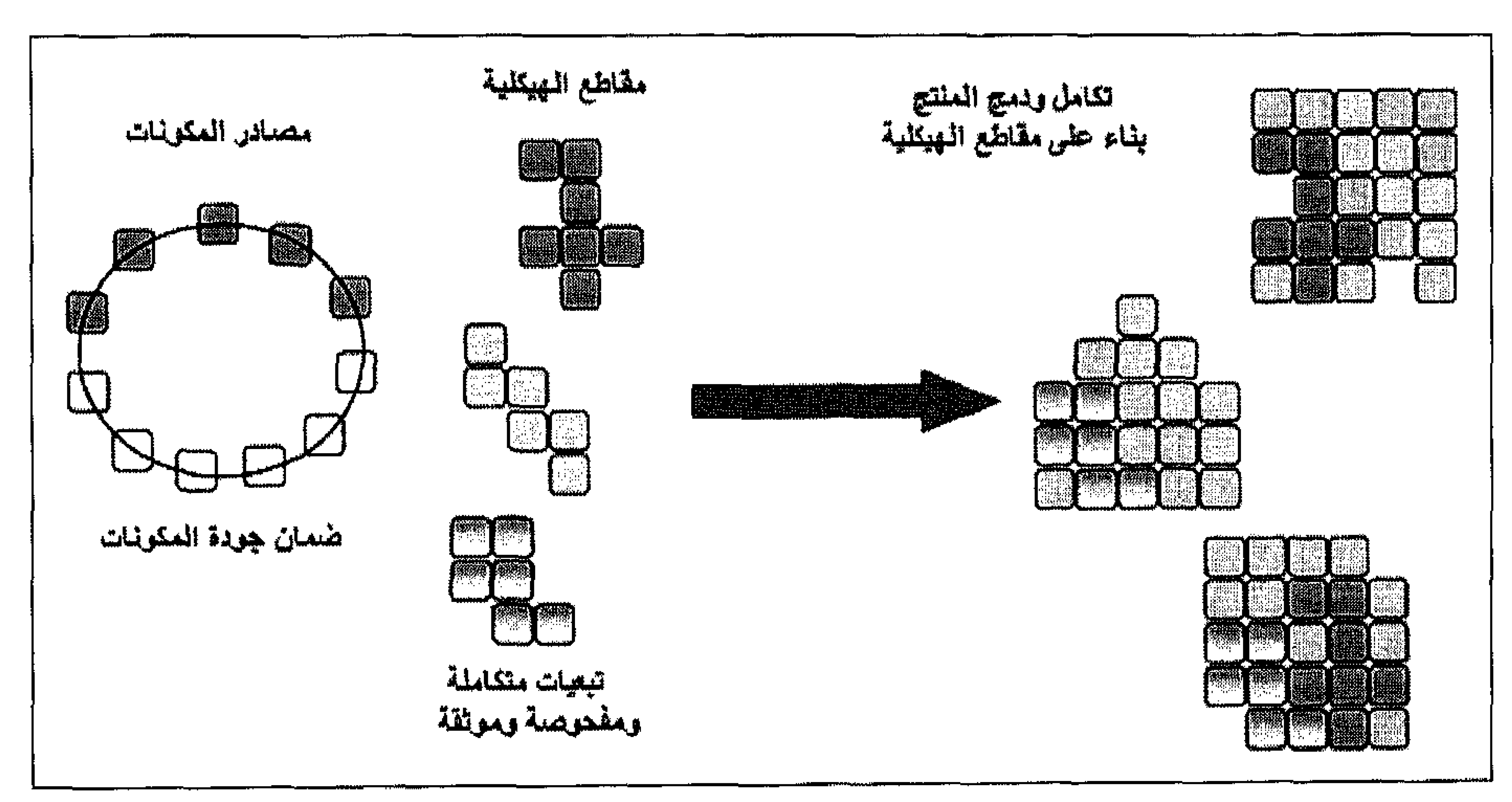
يعرض القسم 2 ـ 4 مجموعة من تحديات البحث التي تواجه تطور ونضج المنهج التركيبي.

2 ـ 2 من المنهجية ذات التوجه التكاملي إلى المنهجية ذات التوجه التركيبي

يعرض هذا الفصل بديلاً من المنهجية مركزية التكامل التقليدية ألا وهو عائلات المنتج. على كلٍّ، قبل أن نناقش ذلك، يجب أن يتم تحديد القضايا المتعلقة بمنهجية المنصة ذات التوجه التكاملي بوضوح أكثر.

تُنظّم عائلات المنتج من طريق الفصل التام بين تنظيم تصميم النطاق وتنظيمات المنتج. يوظف تنظيم تصميم النطاق دورة إصدار دورية يتم فيها إصدار معطيات النطاق بطريقة متكاملة ومُختَبرة، وتعرف دورة الإصدار هذه بمنصة الإطلاق (Platform). تستخدم تنظيمات المنتج منصة الإطلاق كأساس لإنشاء وتطوير المنتج من طريق توسعة المنصة بإضافة خصائص خاصة بالمنتج.

يقسم تنظيم منصة الإطلاق حسب عدد فرق العمل، وفي أفضل الأحوال، يتم عكس هيكلية المنصة على عملية التقسيم. يقوم كل فريق من فرق العمل بتطوير المكوّن (أو مجموعة من المكوّنات المرتبطة) الذي يكون مسؤولاً عن التكامل في منصة الإطلاق وتسليم تنائج التكامل. على الرغم من أن العديد من الشركات قد بدأت تطبق عملية التكامل المستمر بحيث تكون المكوّنات متكاملة ومدمجة باستمرار في أثناء التطوير، إلا أنه عملياً يتم تنفيذ عمليات التحقق والمصادقة في الفترة التي تسبق إصدار منصة الإطلاق، وفي هذه المرحلة يتم العثور على العديد من الأخطاء الحرجة في النظام.



الشكل (2 ـ 1): الطريقة التركيبية

تقوم المؤسسات المسؤولة عن التصميم بتسليم البرامج كنظم برمجية كبيرة متكاملة ومُختبرة مع (API) التي يمكن استخدامها من قبل فرق المنتج لاشتقاق المنتجات منها. في حين تشكل البرامج مجموعة كبيرة من المزايا والصفات مع بعضها بعضاً، فإن تكرار إصدار البرنامج يكون منخفضاً نسبياً مقارنةً بتكرار برامج المنتجات. بناءً على ذلك، تكون المؤسسات المسؤولة عن التصميم

واقعة تحت ضغط لتسليم أكبر قدر ممكن من المزايا والصفات خلال الإصدار. لذا، ثمة توجه إلى العمليات المختصرة خصوصاً عمليات ضمان الجودة، وخصوصاً في أثناء الفترة التي تقود إلى إصدار كبير، يتم تحويل جميع عمليات التحقق والمصادقة إلى فريق التكامل. وحيث إن المكوّنات تفقد الجودة وفرق التكامل تواجه مشكلات التكامل ومشكلات أخرى على مستوى المكوّنات معاً، فإنه تظهر في أسوأ الحالات دورة تكون فيها الأخطاء قد حُدِّدت من قبل فريق الاختبار الذي لا يكون لديه أي فكرة عن هيكلية النظام ويمكنهم تحديد الأعراض فقط، تستلم فرق المكوّنات تقارير عن الأخطاء الموجودة في النظام والتي قد تنشأ من أجزاء أخرى في النظام، وهنا يتوجب على فرق التكامل أن تعمل على إدارة التقارير الواردة من فرق الاختبار والتطوير عن المشكلات الحرجة في النظام، التي قد تؤدي إلى إصدارات جديدة من المكوّنات.

على الرغم من أنه قد بيّنّا العديد من التحديات التي تواجه هندسة البرمجيات التي ترتبط بمنصات البرمجيات، فقد أثبتت المنهجية نجاحاً كبيراً من حيث تحقيق أكبر قدر من كفاءة البحث والتدريب والتكلفة المجدية التي تؤدي إلى منتج غني.

وهكذا، أثبتت المنهجية ذات التكامل المركزي نجاحها في نطاقها الأولى. على كلٍّ، قد يتحول النجاح إلى فشل بسهولة عندما تقرر المؤسسة الاستمرار بناءً على نجاح منصة البرمجية الأولي، وعلى التوسع الكبير في نطاق عائلة المنتج. قد تنتج توسعة النطاق عن قرار المؤسسة جلب المزيد من أصناف المنتج المتواجدة تحت مظلة المنصة أو بسبب قرار التنوع في المنتجات عندما تنخفض تكلفة إنشاء منتجات جديدة انخفاضاً معقولاً. في هذه المرحلة، قمنا بتحديد عدد من الشركات التي توسع نطاق عائلة المنتج البرمجي من دون الحاجة إلى تعديل نمط العملية تعديلاً جوهرياً ما يؤدي إلى عدد من الشؤون الأساسية والمشكلات المنطقية التي لا يمكن تجنبها. على كلٍّ، ونظراً إلى النجاح المبكر الذي حققته المؤسسة، لم يتم تعريف المشكلات كأساس على نحو كافٍ، لكنها عرفت كتحديات تواجه التنفيذ، هذا ولا تتم التغيرات الأساسية على نمط العملية إلا بعد أن تواجه الشركة عواقب مالية كبيرة.

في ما يأتي نعرض هذه المشكلات بالتفصيل، أولاً في ما يختص نطاق المنصة، ومن ثم ما يختص بالانفتاح.

2 ـ 2 ـ 1 مشكلات الإفراط في نطاق العمل

إذاً، ما هي المشكلات التي تسبّب تغيّر نمط العملية التي كانت ناجحة في البداية وتحوله إلى إشكالية؟ في القائمة الآتية، نناقش المشكلات المتعلقة بالنطاق، التي يمكن للمرء ملاحظتها وإدراكها مباشرة.

- عدم توفر عمومية المكوّنات: على الرغم من أن معظم المكوّنات كانت مفيدة لجميع المنتجات في النطاق الأولي لعائلة المنتج، إلا أن عدد المكوّنات المستخدمة في مجموعة جزئية من المنتجات يزداد في النطاق الممتد.

- دمج وظائف غير مكتملة: كما ذكرنا في النقطة السابقة، في النطاق الأولي، تصبح معظم الوظائف مفيدة لمنتج واحد ذات صلة بمنتجات أخرى مع الزمن. بالتالي، هناك اتجاه لدمج الوظائف المختصه بمنتج معيّن مع منصة النظام مبكراً، وقد يكون ذلك في بعض الأحيان قبل استخدام المنتج الأول. عندما يزيد نطاق عائلة المنتج، تصبح مساوىء دمج الوظائف غير المكتملة أكثر وضوحاً.

- التطور البطيء للوظائف: عندما يزيد نطاق عائلة المنتج وعندما تعمل الشركة على المحافظة على المنهجية ذات التوجه التكاملي لتطوير الأجزاء المشتركة من البرمجية، يزيد زمن استجابة منصة النظام تجاوباً لطلبات إضافة وظائف للمزايا الحالية.

- التبعيات الضمنية: في المنهجية ذات التوجه التكاملي، ثمة درجة عالية نسبياً ومقبولة من الترابط بين المكوّنات بسبب وجود بعض المساوئ في هذه المرحلة، كما تزيد إنتاجية مطوّر النظام.
  عندما يتسع نطاق عائلة المنتج، يجب أن يتم تشكيل المكوّنات بتكوينات أكثر إبداعاً، وفجأة تصبح التبعيات الضمنية بين المكوّنات مشكلة حقيقية تعترض إنشاء منتجات جديدة.

- عدم تجاوب تطوير منصة النظام: تكون دورة إصدار منصة النظام البطيئة محبطة، خصوصاً بالنسبة إلى فئات المنتج المبكرة في دورة النضوج. غالباً ما يكون إضافة خاصية جديدة إلى المنتج الجديد مطلوبة بسرعة.

بأيّ حال، تتطلب الخاصية إجراء تغيرات في بعض مكوّنات منصة النظام.

بسبب كون دورة إصدار منصة النظام بطيئة، يكون النظام غير قادر على الاستجابة لطلب فريق عمل تطوير المنتج. يكون فريق العمل مستعداً لتنفيذ هذه الوظيفة بعينها، لكن فريق عمل منصة النظام لا يسمح بذلك بسبب العواقب المحتملة لجودة عمل فريق المنتج.

عند تحليل هذه المشكلات مع النية لفهم الأسباب الكامنة وراءها، يمكن تحديد الأسباب الآتية:

- التقليل من القواسم المشتركة الكاملة: قبل توسيع نطاق عائلة المنتج، تكون منصة النظام قد كونت جوهر وظائف المنتج. بأي حال، بزيادة النطاق، ستتنوع متطلبات المنتجات، وبالتالي يقل مقدار الوظائف المتطلبة لجميع المنتجات سواء بشكل مطلق أو نسبي. بناء على ذلك، يقل عدد المكوّنات المشتركة بين جميع المنتجات، ما يقلل أهمية منصة النظام المشتركة.

- زيادة القواسم المشتركة جزئياً: تزداد الوظائف المشتركة بين بعض المنتجات أو الكثير منها، لكن ليس كلها، بشكل ملحوظ بزيادة النطاق. بناء على ذلك، يزيد عدد المكوّنات المشتركة بين ما بعض المنتجات أو ما بين معظمها. المنهجية النموذجية لهذا النموذج هي تبني التسلسل الهرمي لعائلات المنتج. في هذه الحالة، تقوم مجموعات الأعمال أو فرق العمل المسؤولة عن فئات معيّنة من المنتج ببناء منصة فوق المنصة الشاملة للشركة. على الرغم من ذلك يخفف جزء من المشكلة، إلا أنه لا يوفر آلية فاعلة لمشاركة المكوّنات بين مجموعات الأعمال أو فرق العمل التي تعمل على تطوير المنتجات بفئات مختلفة للمنتج.

- الهيكلية ذات التصميم المُفرط: بزيادة نطاق عائلة المنتج، تتسع أيضاً مجموعة الأعمال والصفات التقنية التي يجب أن تدعمها المنصة المشتركة. على الرغم من أنه لا يحتاج أي منتج دعماً لجميع الصفات، لكن يتطلب من هيكلية المنصة أن تدعمها، وبناء على ذلك، فهي تحتاج إلى تصميم مفرط لتلبية احتياجات جميع المنتجات وفئات المنتج. لكن من شأن ذلك أن يعوق إمكانية التوسع ويزيد من الجهود اللازمة للصيانة.

- الخصائص الشاملة: خصوصاً في النظم المضمنة، تفشل المزايا الجديدة في الالتزام بحدود منصة النظام. ففي حين أن النهج المعتاد هو أنه يتم التمييز بين الخصائص في شيفرة البرمجة الخاصة بالمنتج، إلا أنه غالباً ما تتطلب هذه الخصائص تغييرات في المكوّنات المشتركة أيضاً.

اعتماداً على المجال الذي تطور فيه المؤسسة منتجاتها، فقد تتحول الفكرة العامة للمنصة التي تلتقط الوظائف المشتركة بين جميع المنتجات إلى وهم عندما يزيد نطاق عائلة المنتج

- نضج فئات المنتج: تكون فئات المنتج المختلفة التي تنتجها مؤسسة واحدة في مراحل مختلفة من دورة حياة البرمجة. التحدي هنا هو أنه اعتماداً على مدى نضج فئة المنتج، تختلف المتطلبات في المنصة المشتركة. على سبيل المثال، بالنسبة إلى فئات المنتج الناضجة، تكون الكلفة والموثوقية أهم العوامل، بينما بالنسبة إلى فئات المنتج في طور النضج، تكون غنى المزايا والزمن الذي يمكن توفير المنتج فيه في الأسواق هي العوامل الأهم. يجب توفر منصة مشتركة لتلبية متطلبات جميع فئات المنتج الذي يقود بسهولة إلى توتر بين المنصة وفئات المنتج.

2 ـ 2 ـ 2 مشكلات تتعلق بالمنهجية المغلقة

من التحديات التي تواجه المنهجية ذات التوجه التكاملي التقليدية هو الأهمية المتزايدة لاستخدام البرمجيات التجارية والمكوّنات الجاهزة للاستخدام. الأمر الجوهري هو أن المنهجية ذات التوجه التكاملي تفترض وجود حكم قوي للمتطلبات والمزايا والخطط لكل من المنصة والمنتجات. يقود ذلك تحديداً إلى العديد من المشكلات:

- منصة أساسية غير مرنة: إذا أُدمجت المنصة الأساسية مع برنامج خارجي كالبرمجيات التجارية، لن يكون هناك سيطرة على خطة وعملية التطوير ودورات الإصدار. يؤدي ذلك إلى وجود عدد من القيود ويصبح العثور على هيكلية تُلائم جميع المنتجات المشتقة تحدياً.

- التطور: في حال أثر منتج واحد مشتق في توسيع المنصة عن طريق استخدام برمجية تجارية، فإنه يصعب تضمين هذه البرمجية في منصة المنتج الرئيس لاحقاً.

- عدم تطابق الأدوات والتنظيمات: توظف البرمجيات التجارية منهجيات تنظيمية وأدوات وممارسات للاختبار وضمان الجودة مختلفة، إذ يصعب دمجها في خطوط المنتج البرمجي كما ناقشنا في المرجع[8].

والنتيجة هي أننا بحاجة إلى منهجية أكثر انفتاحاً لا تفترض سيطرة كاملة على المزايا والخطط والأدوات. تفترض المنهجية التركيبية، وكما هو وارد في الأقسام الآتية، وجود بيئة عمل أكثر انفتاحاً وتلبي بسهولة أكثر إعدادات الإدارة غير المركزية والعمليات غير المتجانسة والأدوات.

2 ـ 2 ـ 3 منهجية عائلة المنتج التركيبية

إن الفكرة الأساسية للمنهجية التركيبية لخطوط إنتاج البرمجيات هو أن خط الإنتاج ليس بالحل البرمجي الكامل والمتكامل: لكنه متوفر كبيئة برمجية متكاملة وبيئة أدوات. يتم تولي دور المنصة المرجعية المتكاملة بواسطة مجموعة من الشرائح الهيكلية التي هي عبارة عن تركيبات متكاملة ومختبرة من أحد النظم الفرعية أو أكثر. يتضمن ذلك مكوّنات محددة ذات تماسك عالٍ وهو تكامل عمودي من مكوّنات ذات استقلالية عالية، ويمكن أن توسع لتشمل أطر عمل صغيرة. يجب أن تغطي مجموعة شرائح الهيكلية نطاق خط المنتج البرمجي كاملاً وتمكين تكامل المنتج لاحقاً. بينما يظهر مفهوم شرائح الهيكلية مماثلاً لعمل ذي علاقة في نمذجة الهيكلية[9, 11, 15]، فإن فكرتنا العامة عن شريحة الهيكلية ترتكز على التكامل والاختبار، ونحن نركز أكثر على منهجية خط إنتاج البرمجية الكامل.

على سبيل المثال، في الهاتف الخلوي، يمكن أن تكون مجموعة من تطبيقات الوسائط المتعددة التي تستخدم كاميرا ذاتية عبارة عن شريحة هيكلية. قد يتضمن ذلك العديد من متحكمات الكاميرات المتبادلة وتطبيقات التقاط الصور وعرضها والعديد من مكوّنات نظم التشغيل الخاصة بتخزين بيانات الوسائط واسترجاعها بسرعة. قد تختلف هذه المكوّنات اعتماداً على درجة جودة الكاميرا وعدد الكاميرات في الهاتف. إضافة إلى ذلك، قد يعتمد دعم نظام التشغيل للكاميرات على الجودة (Resolution) والأداء المطلوب في تخزين البيانات. هذا ويجب أيضاً تحديد الاعتمادية على مكوّنات شرائح الهيكلية الأخرى. في هذا المثال، قد يكون تطبيق عرض الصور مرتبطاً مع تطبيق الرسائل وذلك لإرسال الرسائل مباشرة. إضافة إلى ذلك، قد يعتمد تطبيق

المؤتمرات المصورة بالفيديو على دعم الكاميرا لتسجيل المكالمات الفيديوية. ففي حين أن التبعيات يجب أن تكون موثقة ويمكن اختبارها باستخدام عيّنة من الشيفرة البرمجية، إلا أن إنشاء المنتج الفعلي هو ما سيتم تكامله. يمكن قراءة المزيد من التفاصيل عن شرائح الهيكلية في الأقسام التالية.

ملخص ذلك، أن عائلة المنتج البرمجي التركيبية تتضمن ما يأتي:

- مجموعة من المكوّنات المتماسكة التي تسهّل التكامل.
- مخططات الهيكلية الشاملة والمبادئ التي تقود إلى تطوير المنتج لاحقاً.
- شرائح الهيكلية التي تغطي نطاق خط إنتاج البرمجية الكامل وتجسد تكامل المنتج.
- سيناريوهات الاختبار وبيئات الفحص الخاصة للمكوّنات وشرائح الهيكلية.
- التوثيق التفصيلي عن الاعتمادية بين المكوّنات وشرائح الهيكلية وتبعياتها والتركيبات الموصى بها.

بناءً على البيئة المرنة والمفتوحة، يتم تكوين منتجات كاملة ومتكاملة ومختبرة.

إن باستطاعة منهجية عائلة المنتج التركيبية أن تتضمن منصة متكاملة لمنتجات أساسية معيّنة بدون دمج جميع مكوّناتها. قد يخدم ذلك أيضاً منتجاً مرجعياً يستخدم للاختبار والمكوّنات غير الوظيفية.

أما الفارق الرئيس فهو أن أغلب ما يشتق من ذلك المنتج هو عبارة عن تركيب من الأجزاء الموضحة في النقاط السابقة.

في المنهجية ذات التوجه التكاملي، يجب أن تحل حزمة البرمجية المتكاملة جميع التبعيات والتقاطعات بين المكوّنات بواسطة الشيفرة البرمجية الفعلية. في هذه المنهجية، يتم ذلك عن طريق دمج مجموعة من المكوّنات بواسطة شرائح هيكلية وعن طريق توثيق التبعيات الخارجية. إن توثيق التبعيات أمر في غاية الأهمية ويجب أن يتم كجزء أساسي في المنهجية التركيبية. مقارنة بالمنهجية التقليدية، فإننا نقوم بجعل هذه التبعيات واضحة ونترك الدمج الفعلي لعملية إنشاء المنتج.

أما الميزات الرئيسة والمواصفات التقنية لهذه المنهجية فهي:

● **هيكلية المنتج أكثر مرونة**: لقد عُرّفت هيكليات المنتجات البرمجية بطريقة أكثر مرونة وليست مقيدة بهيكلية المنصة. وسبب ذلك عدم وجود هيكلية منصة مفروضة، على الرغم من أنه قد تتوفر هيكلية مرجعية تدعم هيكلية المنتج.

● **المسؤولية المحلية**: تتقبل المكوّنات المعادة الاستخدام التي قد تستخدم في إنشاء المنتج مستوى أكبر بكثير من المسؤولية المحلية. وهذه المسؤولية تعني أن كل مكوّن من المكوّنات (أ) يستخدم واجهات استخدام محددة ومهيئة مسبقاً ومتطلبة، (ب) يتحقق من أن جميع واجهات الاستخدام مرتبطة بشكل صحيح في فترتي التركيب والنشر، وأن المكوّن يمكن أن يوفر سيناريوهات الاستخدام أو المزايا (أو جزء منها) التي يعد بها، (ج) يتضمن ذكاء كافياً ليضبط نفسه ديناميكياً لتتواءم مع مكوّنات الأجهزة الخادمة أو التابعة التي تعرض وظائف أقل من المتوقع أو تتطلب وظائف أكثر من المتوقع.

● **تقليل كلفة الدمج**: تتم عملية دمج مكوّنات البرمجيات معادة الاستخدام والخاصة بمنتج بعينه بواسطة ذلك المنتج، ولا تنفذ أي عمليات دمج للمنصة. ونظراً إلى الذكاء المتنامي للمكوّنات، انخفضت الجهود المبذولة لتحقيق الدمج لتصبح جزءاً يسيراً من الجهد المبذول في المتطلبات.

من الواضح أن هذه المنهجية تؤدي إلى وجود العديد من التحديات التي تهتم بجميع جوانب عملية تطوير البرمجيات. سنناقش هذه التحديات في القسم 2 ـ 4. لكننا سنناقش أولاً تأثير هذه المنهجية في جوانب تطوير البرمجيات المختلفة.

2 ـ 2 ـ 4 الفروقات الأساسية في المنهجية التركيبية

حتى نشرح المنهجية التركيبية بشكل أدق وعلاقتها بالمنهجية ذات التوجه التكاملي، ناقشنا هاتين المنهجيتين من خمس وجهات نظر، نعتقد أنها ذات أهمية سائدة في سياق توسيع نطاق عائلات المنتج البرمجي، وهي: استراتيجية الأعمال، الهيكلية، المكوّنات، إنشاء المنتج والتطور.

● **استراتيجية الأعمال**: غالباً ما يكون السبب الأصلي لتبني عائلات المنتج هو تقليل نفقات التدريب والتطوير من خلال مشاركة معطيات البرمجية بين عدة

منتجات. على الرغم من أن ذلك لا يعني إهمال التدريب والتطوير في المنهجية التركيبية، بل يكون التركيز الأساسي على زيادة نطاق عائلة المنتج إلى الحد الأقصى. على الرغم من أن تكلفة التدريب والتطوير والزمن اللازم لإخراج المنتج إلى السوق عوامل ذات صلة في أي مؤسسة موجهة تكنولوجياً، إلا أن الأساس المنطقي الأكثر أهمية للمنهجية ذات التوجه التركيبي هو إعطاء مطوّري المنتج مرونة وحرية أكثر، وبذا تصبح عملية إنشاء مجموعة أوسع من المنتجات أكثر سهولة.

- **الهيكلية**: مبدئياً، يتم تحديد هيكلية عائلة المنتج كهيكلية بنيوية كاملة في ما يتعلق بالمكوّنات والروابط. تكون الهيكلية واحدة لجميع المنتجات أما الاختلاف فيُحدد من خلال نقاط التباين في المكوّنات. عند الانتقال إلى المنهجية التركيبية، يقل التركيز على العرض البنيوي للهيكلية بينما يزيد التركيز على القواعد الهيكلية المبطنة ما يضمن قابليتها للتركيب. كما ناقشنا مسبقاً، الفرق الأساسي بين هذه المنهجية والمنهجيات الأخرى فيتمثل في أن الهيكلية ليست موصوفة من حيث المكوّنات والروابط، بل من حيث الشرائح الهيكلية وقواعد التصميم وقيود التصميم.

- **المكوّنات**: في أثناء المرحلة الأولى من عائلة المنتج البرمجي، يتم تنفيذ المكوّنات لهيكلية معيّنة محددة في جميع جوانبها أو معظمها. تتضمن المكوّنات نقاط تباين لتلبي الفروقات بين المنتجات المختلفة في العائلة، لكنها لا تمتد إلى ما وراء واجهات المكوّنات على نحو ذي أهمية. أخيراً، تنفّذ المكوّنات بطريقة تجعلها تعتمد على تنفيذ مكوّنات أخرى بدلاً من الواجهات المحددة والضمنية. عندما يحدث تطور نحو اعتماد المنهجية التركيبية، يبقى التركيز منصباً على المكوّنات. ومع ذلك، لم يتم تطوير هذه المكوّنات بطريقة مخصصة، بل أنها مقيدة في التنفيذ من قبل الهيكلية ـ أي من قبل مقاطع الهيكلية وقواعدها وقيودها التي نوقشت مسبقاً. عند دمج كل مكوّن من المكوّنات في شريحة هيكلية واحدة أو أكثر، يمكن ضمان قابلية التركيب لهذه الشرائح.

- **إنشاء المنتج**: يفترض النموذج ذو التوجه التكاملي أن المنصة المدمجة مسبقاً التي تتضمن وظيفة عامة تكون متطلبة من جميع المنتجات أو معظمها من منتجات العائلة. يتم إنشاء المنتج من طريق استخدام منصة مدمجة مسبقاً

كأساس، ومن ثم إضافة شيفرة برمجية مخصصة للمنتج فوق المنصة. وعلى الرغم من أن ذلك ليس ضرورياً، إلا أن الشركة تكون منظمة ضمن هذه الحدود. تعمل المنهجية جيداً عندما تكون عائلات المنتج ضيقة النطاق، لكن فعاليتها تكون أقل جودة عندما يكون نطاق عائلة المنتج واسعاً. في المنهجية التكاملية، يكون الهدف الضمني تسهيل اشتقاق مدى المنتجات الواسع الذي قد يحتاج إلى تشكيل المكوّنات بمدى واسع متساوٍ من عمليات التهيئة. وعليه فإن إنشاء المنتج يتم باختيار معظم المكوّنات الملائمة وتهيئتها حسب متطلبات المنتج، وتطوير شيفرة البرمجة حيثما يلزم ضبط التفاعل بين المكوّنات بالنسبة إلى متطلبات محددة للمنتج، وتطوير شيفرة برمجية خاصة بالمنتج فوق المكوّنات التي أعيد استخدامها.

● **التطور**: في المنهجية ذات التوجه التكاملي، غالباً ما يكون هناك نزعة تفضيلية قوية نحو دمج مزايا ومتطلبات جديدة في المنصة التي سبق دمجها. وسبب ذلك أنه يجب إضافة مزايا جديدة قبل موعد الاستحقاق لجميع المنتجات، وبالتالي يكون الأسلوب الأكثر فعالية من حيث التكلفة هو تنفيذ ذلك مباشرة من دون تأجيل. كبديل، في حال كانت المنهجية المَنْوي استخدامها تتطور من حيث الوظائف المخصصة للمنتج وتصبح سلعاً بمرور الوقت، ويتم دمجها في المنصة فإنها تضعف. في المنهجية التركيبية، تكون فرق العمل الخاصة بالمنتج وفرق العمل الخاصة بالمكوّنات مسؤولة عن تطور قاعدة الشيفرة البرمجية. تعمل فرق العمل الخاصة بالمنتج على زيادة المكوّنات المتوفرة بإضافة وظائف يتطلبها المنتج خاصتهم، لكن يتم الحكم على مدى منفعتها وجدواها للمنتجات المستقبلية كذلك. هذا وقد تعمل فرق عمل المنتج على إنشاء مكوّنات جديدة للغرض نفسه. أما فرق العمل الخاصة بالمكوّنات، إن استخدمت، فتُعنى أكثر بإضافة المزايا المتطلبة لعدد من المنتجات. مثال نموذجي على ذلك تنفيذ إصدار جديد لبروتوكول الاتصال.

2 ـ 3 المكوّنات والشرائح الهيكلية

في ما تقدم، ناقشنا المنهجية ذات التوجه التركيبي. في هذا القسم، نتوسع في موضوع تطوير البرمجيات التركيبية بناء على المقومات الرئيسة والمكوّنات وشرائح الهيكلية. الطبيعة غير المركزية للتطوير التركيبي يجعل جمع أنماط تطوير عديدة ممكناً لأن المعطيات المهيأة يتم تطويرها بالضرورة بشكل مستقل.

2 ـ 3 ـ 1 تقنية المكوّنات

الفرق الأساسي بين التطوير ذي التوجه التكاملي والتطوير ذي التوجه التركيبي أن المنهجية التركيبية غير مركزية. أما المكوّنات التي يتم تكوينها فيتم تطويرها بشكل مستقل عن بعضها البعض، وعن المنتجات التي تستخدم هذه المكوّنات فيها. أفضل تعريف للمكوّن ملائم لهذا الفصل قدمه كليمنس زايبيرسكي (Clemens Szyperski) في كتابه عن المكوّنات «المكوّن البرمجي هو وحدة من تركيب ذي واجهات محددة تعاقدياً وتبعيات واضحة السياق فقط. المكوّن البرمجي يمكن نشره بشكل مستقل ويكون عرضة للتركيب من قبل أطراف خارجية»[19].

بكلمات أخرى، يتم تطوير المكوّنات بصورة مستقلة، ومن ثم يتم دمجها وتكاملها من قبل طرف خارجي. نحن نفسر العبارة «وحدة من تركيب» بشكل واسع هنا. فحسب خبرتنا، واعتماداً على المستوى النظري، تقسم النظم البرمجية الكبيرة جداً إلى مئات المكوّنات التي قد تكون بدورها كبيرة جداً.

كما هو ملاحظ في المقدمة، النشاط الأساسي في المنهجية ذات التوجه التكاملي هو مرحلة الدمج والتكامل التي يتم خلالها تحديد الأخطاء والمشكلات الخطيرة وإصلاحها. تتيح المنهجية التكاملية في تطوير البرمجيات لمطوّر المنتجات والمكوّنات تشغيلها بشكل مستقل. وبالتالي فهي منهجية قابلة للقياس تسمح لمزيد من المطوّرين: (أ) أن يعملوا معاً (عن طريق جعل عملهم غير مركزي) و(ب) بناء منتجات برمجية أكبر (عن طريق جمع مخرجاتها).

لكن يصعب إنجاز عملية اختبار التكامل عند تطوير النظام بالطريقة التركيبية بسبب الأنماط العملية المستقلة لفرق تطوير المكوّنات والمنتج وغياب التخطيط المركزي وصنع القرار.

طبعاً، يجب أن يقوم مطوّر المنتج باختبار تهيئة المكوّنات المستخدمة في المنتج أو أي منتج آخر ذي علاقة. يجب أن لا يغير مطوّرو المنتج المكوّنات من حيث التهيئة (التلاعب بها جانباً باستخدام API الخاص بالتهيئة). كما يصعب على مطوّري المنتج أن يطلبوا من مطوّري المكوّنات أن يصلحوا الأخطاء والأعطال، وأن ينفذوا المزايا أو يغيروها وغير ذلك. قد يكون ذلك صعباً لعدة أسباب:

- يتوفر المكوّن عن طريق مطوّري نظم من خارج الشركة (مثلاً، تعاقد جزئي أو مورّد مكوّنات تجارية جاهزة للاستخدام أو مشروع تجاري). قد يكون لهؤلاء الأطراف بعض الالتزامات تجاه توفير وصيانة منتجاتهم (مثلاً، المنتجات التي يحكمها عقد دعم) لكن قد لا يبدون اهتماماً كبيراً بقضايا خاصة بالمنتج تظهر في أثناء دمج المنتج.

- يتم تطوير المكوّن حسب خريطة الطريق الموضوعة له في الإصدارات الكبيرة والصغيرة المخطط لها. أما القضايا التي لا تلائم خارطة الطريق تلك، فمن غير المرجح أن تأخذ اهتماماً خلال فترة قريبة.

- التغيرات المطلوبة التي تتعارض مع تلك التي يتطلبها مستخدمون آخرون قد تستخدم مكوّنات معيّنة في عدة منتجات. هذا وقد تكون المكوّنات الخارجية المستخدمة استخدمت في منتجات منافسة. عند مواجهة مثل هذه التعارضات، يجب أن يتم توفير الحل في المنتج بدلاً من حل مشكلة المكوّن.

بناء على ذلك، يجب أن تكون المكوّنات المستخدمة في المنتج ثابتة ومختبرة جيداً قبل أن يصبح المنتج في مرحلة الدمج والتكامل. بمعنى آخر، يتضمن تبنّي المنهجية التركيبية وضع تركيز أكثر على اختبار المكوّن وجعل ذلك مسؤولية فرق تطوير المكوّن.

في حال لم يتم توفير الوظيفة الأساسية لمنتج معيّن من قبل بيئة المكوّن بحيث لا يمكن إضافتها في شيفرة برمجية لاحقاً، فقد يتطلب الأمر إضافة مكوّن مشتق. يحدث ذلك في المنهجية التركيبية أكثر من غيرها إذ لا تكون خريطة طريق والمزايا منسقة جيداً. نحن نرى أن هذا الخيار ميزة إذ إنه يزيد المكوّن الداخلي. باستخدام المنهجية التركيبية للتطوير، فقد يتوقع أن تكون هذه المكوّنات قابلة للاستخدام في منتجات أخرى مطوّرة في الشركة نفسها.

2 ـ 3 ـ 2 الشرائح الهيكلية والمكوّنات

تُعنون المنهجية التي شرحناها في هذا القسم عملية فحص دمج وتكامل المكوّن باستخدام مفهوم الشرائح الهيكلية والدمج الجزئي. الشريحة الهيكلية لا تصف هيكلية المنتج كاملة، بل تصف المظاهر المرتبطة ببيئة المكوّن المتوقع أن يستخدم فيها. تعرّف المواصفة IEEE 1471 الهيكلية البرمجية على أنها

«التنظيم الأساسي لنظام منصوص عليه بمكوّناته، والعلاقات التي تربط بين بعضها البعض والبيئة، والمبادئ التي توجّه تصميمها وتطورها»[10]. في حالة التطوير التركيبي، فإنه ليس من المستحيل على مطوّري المكوّن اعتبار جميع المنتجات التي تستخدم مكوّناتهم. إضافة إلى ذلك، قد تحتوي هذه المنتجات على بعض العوامل المشتركة أكثر من حقيقة أنها تستخدم المكوّن. بمعنى آخر، بدلاً من التركيز على الهيكلية الكاملة، يجب أن يركّز مطوّرو المكوّن على جزء منها، ذلك الذي يرتبط مباشرة بالمكوّن. ونسمي ذلك، الشريحة الهيكلية.

الشريحة الهيكلية هي مجموعة من المكوّنات المقترنة بقوة، ينصح باستخدامها بهذا التركيب في المنتجات. في معظم الحالات، يمثل ذلك نظاماً فرعياً للهيكلية. مثلاً، في الهواتف النقالة، تكون الشريحة الهيكلية عبارة عن مجموعة من تطبيقات الوسائط المتعددة المعدة لكاميرا الهاتف وتتضمن متحكمات أو مجموعة من المكوّنات التي تعمل على تشغيل ملفات الوسائط. كما تعرّف الشريحة الهيكلية تبعياتها الخارجية لمكوّنات أخرى أو نظم فرعية أخرى. لا تكون الشريحة الهيكلية مكتملة بدون هذه التبعيات الخارجية ويجب أن تصف علاقتها مع هذه النظم الفرعية الخارجية. بمعنى آخر، تتضمن الشرائح الهيكلية افتراضات عن كيفية ارتباط النظم الفرعية الأخرى بها.

2 ـ 3 ـ 3 الشرائح الهيكلية واختبار التكامل

كما ناقشنا مسبقاً، يشكل اختبار تكامل شرائح الهيكلية جزءاً أساسياً في المنهجية التركيبية. لهذا، إحدى أهم القضايا أنه يجب تحديد التبعيات للمكوّنات الخارجية. لاختبار مكوّن أو شريحة هيكيلة، يجب أن يوفّر المطوّر بيئة تفي بهذه التبعيات. فباستخدام تهيئة الشريحة الهيكلية، يمكن تنفيذ تطبيقات بسيطة تختبر الجوانب المختلفة لوظيفة المكوّن. في حال كان هناك نية لاستخدام امتداد بإضافة مكوّن آخر، يكون إنشاء هذا الامتداد طريقة مفضلة للاختبار.

يمكن تصنيف التبعيات للمكوّنات الخارجية في مجموعتين:

● **تبعيات الاستخدام**: يعتمد المكوّن على غيره من المكوّنات. وقد يكون ذلك إما إصدارات محددة من المكوّنات أو كتطبيقات (API) قياسية محددة، إذ أصبح ذلك أمراً أكثر شيوعاً في عالم جافا (Java).

● **استخدام التبعيات**: تشير هذه التبعيات إلى المكوّنات الأخرى التي

تعتمد على هذا المكوّن أو التي يجب أن تعمل مع هذا المكوّن بشكل صحيح.

قد تكون العلاقات بين التبعيات مقترنة أيضاً، مثلاً، إن استخدام المكوّن A يقتضي أيضاً استخدام المكوّن B. يجب أن تكون مثل هذه العلاقات موثقة بالطبع. تتم عملية اختبار التكامل طبيعياً كجزء من تطوير المنتج. كما أشرنا مسبقاً، يكون تحديد الأمور المتعلقة بالمكوّنات متأخراً جداً بشكل عام. وبناء عليه، يجب أن يتم اختبار التكامل مبكراً عند مستوى المكوّن وشريحة الهيكلية. فبينما يكون من المستحيل تحقيق منتجات كاملة كجزء من عملية اختبار المكوّن، يكون من المجدي توفير بيئة تستخدم للاختبار عيّنة من الاستخدامات المعروفة أو المتوقعة للمكوّن في المنتجات الفعلية.

- بالنسبة إلى تبعيات الاستخدامات (Uses Dependencies)، قد تستخدم اختيارات من المكوّنات التي قد يستخدمها مطوّرو المنتجات. إذا كانت التوافقية أمراً مهماً، فقد يتم إنشاء تهيئات متنوعة لشريحة الهيكلية بمكوّنات مختلفة، مثلاً، اختبر إصدارات متنوعة من المكوّن نفسه أو تطبيقات مختلفة ل API نفسه. عندما يوسع المكوّن مكوّناً آخر، يمكن وصف العلاقة بالمكوّن الموسع لعلاقة استخدام أيضاً. في هذه الحالة، يكون الاختيار سهلاً نظراً إلى كونه المكوّن الذي يتم توسعته هو دائماً نفسه.

- قد تكون تبعيات الاستخدام (Usage Dependencies) أكثر صعوبة. ففي بعض الحالات، قد يكون من الاستحالة بمكان الاختبار باستخدام تهيئة المنتج كاملة. عند تطوير إصدار جديد من المكوّن (تحديداً، عندما يتضمن ذلك تطوير التغيرات في واجهة برمجة التطبيقات API)، فمن غير المرجح أن تكون المكوّنات الموجودة متوافقة. في هذه الحالة، لابدّ من محاكاة تبعيات الاستخدام باستخدام عمليات تنفيذ وهمية. إضافة إلى ذلك، في حالة استخدام المنتجات التجارية، قد لا يكون المنتج البرمجي متوفراً بالكامل لمطوّر المكوّن.

2 ـ 3 ـ 4 تبعيات المكوّن

في حين أن اختبار كافة المجموعات المتكونة من جميع التبعيات أمر مستحيل عموماً، إلا أنه يمكن أن نقرر مجموعات المكوّنات المعروف لنا أنها ستعمل كما هو متوقع. يمكن توفير هذه المعلومات في وثائق النظام.

تبدو هذه الممارسة مألوفة في صناعة البرمجيات. على سبيل المثال، تنص

ملاحظات الإصدار ذي النسخة 1.1.0 من (Apache's Jakarta Commons Logging component) على ما يلي:

«يتم تجميع كافة الأنواع (Classes) المركزية باستخدام 1.2.x JDK، وقد يعمل JCL على بعض 1.1 Series JREs المزادة، لكن يوصى أولئك الذين يرغبون في تنفيذ هذه الأنواع على 1.1 JREs بتحميل المصدر وإنشاء تطبيقات مخصصة منه». هذا مثال جيد على التبعية التي قد تعمل، لكن لا يوصى بها لأن مطوّري المكوّن لا يضمنونها في إجراءات الاختبار الخاصة بهم. تشير التوصية بوضوح إلى أنه لا يحبذ أن يستخدم المطوّرون 1.1 JRE لكن يمكنهم ذلك إن رغبوا. وهذا مثال أيضاً حيث من الممكن أن يختار مطوّرو المنتج إنشاء إصداراتهم الخاصة من المكوّنات على مسؤوليتهم الخاصة.

إن توثيق مجموعات العناصر التي تعمل والموصى بها أمر مألوف. هذا وسيشهد العديد من مزودي المكوّنات بأن برمجياتهم تعمل في مجموعات مع مكوّنات أخرى معيّنة، وستكون قادرة على توفير دعم واسع للمستخدمين إذا استخدموا المكوّنات الموصى بها في مجموعات مع إصدارات المكوّنات التي ينتجونها. إن هذه الشهادة ونموذج الدعم هما الأساس لمعظم مجموعات البرمجيات الجاهزة للاستخدام كـ MySQL وJBoss.

عملياً، يدفع ذلك المطوّرين (وبشدة) إلى تفضيل مكوّنات جودة الإصدار على إصدارات التطوير (إن توفرت) ولتلبية أي تبعيات لهذه المكوّنات باستخدام المكوّنات الموصى بها. إن قيام المطوّرين بذلك يتيح لهم الاعتماد على الاختبارات التي نفذها مطوّرو المكوّن مسبقاً والتركيز أكثر على اختبار تفاصيل المنتج.

النتيجة الثانية هي أن ذلك يجعل من تحقيق التبعيات عملية قرار يتم إكمالها مبكراً في عملية تطوير المنتج. بشكل عام، في المنتج الجديد، تتحقق معظم التبعيات في أثناء مرحلة تصميم هيكلية المنتج. هذا وقد تحدث بعض عمليات الترقية لإصدارات جديدة، لكن ليس من المرجح أن يرغب مدراء المنتج بالمخاطرة بعد أن يصبح جزءاً من عملية اختبار تكامل المكوّن. تختلف هاتان الممارستان عن المشاركة في تطوير المكوّنات والمنتجات في خط إنتاج البرمجيات. في أثناء مرحلة التكامل، يتم دمج المكوّنات باستمرار في إصدارات تطوير مكوّنات أخرى. بطريقة مماثلة، سينتهي الأمر بمطوّري المنتج باستخدام

اصدارات معدلة من المكوّنات، ما يلغي بعض جهود اختبار التكامل التي نفذت مبكراً. الممارسات المشار إليها أعلاه لا تشجع حدوث ذلك وتتيح لمطوّري المنتج البناء مستفيدين من الأساس المختبر جيداً.

2 ـ 3 ـ 5 أمثلة

في هذا القسم، لقد وصفنا كيف يقوم مطوّرو المكوّنات بعمل مستوى الدمج والتكامل من دون بناء منتج كامل بناءً على مكوّناتها. هذا الأمر مهم لإنشاء مجموعات المكوّنات لأنها تتيح لمطوّري المنتج الاعتماد على اختبار التكامل الذي تم انجازه، بدلاً من إجراء الاختبار بأنفسهم.

ثمة مثال مقنع على ذلك وهو نظام توزيع لينوكس Debian(*)، وهو مجموعة من آلاف الحزم البرمجية مفتوحة المصدر(**) والتي تعمل على لينوكس. إن هدف تأسيس Debian المعلن هو توفير توزيع تكاملي ثابت ومختبر بالكامل. أساسياً، يتكون معظم العمل من دمج هذا العدد الكبير من الحزم في التوزيع. ففي حين أن هناك بعض التطوير الخاص بـ Debian، إلا أن معظم ذلك يتكون من البنية التحتية الخاصة بـ Debian والشيفرة البرمجية خاصتها. إضافة إلى ذلك، يتم نشر التغذية الراجعة عن اختبار التكامل في الحزم التابعة مفتوحة المصدر، ويترافق ذلك في الأغلب مع الإصدارات الصغيرة الملحقة التي توفر حلولاً لمشكلات معيّنة (Patches).

في المرجع[1]، قدّمت بعض الاستراتيجيات المؤثرة في ما يتعلق بحجم التوزيع: بالقياس، وجد أن الإصدار 3.1 (a.k.a. Sarge) يتكون من 230 مليون سطر من الشيفرة البرمجية MLOC. أما الإصدار 3.0 الذي صدر قبل ثلاث سنوات فقط من الإصدار 3.1 فقط وجد أنه يتكون من 105 مليون سطر من

(*) هو نظام تشغيل متكامل، بدأ كإصدارة لجنو/لينوكس وأصبح اليوم أساساً للعديد من التوزيعات الأخرى من لينوكس، كما إنه يدعم أنوية نظم تشغيل غير لينوكس مثل نواة هيرد، ونواة فري بي.إس.دي ضمن مشروع Debian GNU/kFree BSD ويتم تطوير إصدارة Nexenta بنواة Open Solaris عليه، ولكنها ليست جزءاً من مشروع Debian بشكل رسمي، ودبيان هو مشروع غير ربحي على الإطلاق، وهو لا ينتمي لأي شركة، طوره الكثير من المبرجين من كافة أنحاء العالم (المترجم).

(**) البرامج مفتوحة المصدر (Open Source Software) هي البرامج التي يمكن تعديل شيفرتها البرمجية وهي أكثر مرونة للمستخدم من البرامج الأخرى التي لا يمكن التعويل عليها، وهذه البرامج قد تكون مجانية أو بمقابل مادي (المترجم).

الشيفرة البرمجي، بينما تكون الإصدار 2.1 (حسب المرجع)[20] من 55 مليون سطر من الشيفرة البرمجية. وهذا يعني أن التوزيع قد تضاعف أربع مرات في حوالى خمس سنوات. حسب (Wheeler) الذي طبق نظام COCOMO على هذه المقاييس، يتوافق ذلك مع استثمارات بلغت قيمتها بلايين الدولارات ما تطلب آلاف مهندسي البرمجيات للعمل معاً في إطار زمني يوسع الزمن الفعلي المستغرق لتسليم هذه الإصدارات من Debian الذي تحكمه شركة صغيرة تمولها المنح المالية من القطاع الصناعي والأفراد. وهذا يعني أن منهجية التكامل التام غير ملائمة لنماذج COCOMO لتنتج نظاماً برمجياً مقارنة بـ Debian من حيث حجم الشيفرة البرمجية.

من عمليات التطوير الممتعة التي نشأت في السنوات الأخيرة نشوء المشاريع مفتوحة المصدر، حيث يتم تطوير البرمجية بتعاون المطوّرين الذين يعملون لدى شركات منافسة أو يحصلون على الدعم منها. ثمة دافع وراء هذا التوجه، ألا وهو المقاييس المذكورة أعلاه: فهي الطريقة الوحيدة الفاعلة من حيث التكلفة لتطوير حزم برمجية كبيرة بالاستعانة بالعديد من مهندسي البرمجيات في إطار زمني قصير.

إذا لم يكن النظام يميز بين المنتجات الأساسية، فإن استمرار العمل تحت مظلة ترخيص النظم مفتوحة المصادر قد يجعلها تميز بين المنتجات أكثر. إن توجه الشركات الطبيعي لحماية استثماراتها وملكياتها الفكرية يسبّب تعارضاً مباشراً مع هذا الأمر ويعرضها لتحديات تنظيمية واستراتيجية. من الأمثلة الجيدة على الشركات التي تغلبت على هذه المعارضة شركة IBM. فمنذ خمس سنوات خلت، أصدرت الشركة نظام Eclipse لبيئات التطوير بلغة الجافا الذي يعمل ضمن ترخيص مفتوح المصدر. ثم حولوا عمليات تطوير وملكية المنتج إلى شركة مستقلة. أفادت الشركة من ذلك إفادة تفوق خسائر المبيعات التي تكبدتها الشركة من المنتج Visual Age. أولاً، العديد من المنافسين شاركوا في المشاريع، ما أدى إلى نشوء بيئة تطوير تعتبر منصة تطوير معيارية في صناعة البرمجيات. نظراً إلى أن IBM مرتبطة ارتباطاً وثيقاً بمنتج Eclipse، فإن ذلك يعني أن الشركات الأخرى قد فقدت القدرة على تمييز المنتجات، في حين أن IBM قد اكتسب بعض تلك القدرة.

إضافة إلى ذلك، ففي حين تستمر IBM في استثمار مصادر كثيرة في

Eclipse، فإن نسبة كبيرة من هذه الاستثمارات مشتركة مع المنافسين. إذاً إما أن يكونوا قد اقتطعوا بعض التكاليف أو أنهم استطاعوا التحسّن والابتكار بتكاليف أقل (مقارنة بإنجاز كل شيء في الشركة). كما أن بعض الأمور التي لم تكن IBM لتهتم لها فقد تم تمويلها من قبل جهات أخرى وهي الآن قيد الاستخدام من قبل عملاء IBM. أخيراً، إن مساهمة IBM القوية في هذا المنتج تجعل من الشركة شريكاً مهماً للمنتجات والخدمات ذات الصلة كالوسائط والمعدات والخدمات الاستشارية التي توفرها IBM. يمكن الاطلاع على النقاشات والجدال حول هذا الموضوع بالرجوع إلى Cathedral and Bazaar[17].

2 ـ 4 معوقات بحثية لطريقة المعالجة التركيبية

تمثل المنهجية التركيبية تطوراً محتملاً للشركات التي تستخدم منهجية خط المنتج البرمجي لتطوير برمجياتها. على كلٍّ، ثمة تحديات يجب تحديدها. في هذا القسم، نهدف إلى توفير نظرة عامة عن هذه التحديات.

2 ـ 4 ـ 1 إدارة المتطلبات اللامركزية

إن نتيجة استخدام المنهجية التركيبية أن المتطلبات اللازمة للمنتجات المتكاملة يتم إدارتها بشكل منفصل عن المكوّنات المفردة التي يستخدمها النظام.

في تطوير خط إنتاج البرمجيات ذات التوجه التكاملي، يتم إدارة المتطلبات بشكل مركزي. عند تطوير منتج جديد بناء على برمجية خط المنتج، يعرّف مصمم المنتج المتطلبات الخاصة بذلك المنتج والمتطلبات التي يحققها خط إنتاج المنتج. إن تطور خط المنتج بدوره محكوم مركزياً وتوجِهه المكوّنات المشتركة بين المنتجات ومتطلبات أخرى يعتقد أنها مفيدة للمنتجات المستقبلية. أما تطوير المكوّنات فردية فيكون محكوماً بالمتطلبات المدارة مركزياً.

من سمات المنهجية التركيبية أنه لا توجد إدارة مركزية للمتطلبات والمزايا. يختار مطوّرو المنتج المكوّنات وشرائح الهيكلية بناء على كيفية دعمهم لمتطلبات المنتج، لكنهم قد يأخذون بالاعتبار عوامل أخرى، منها على سبيل المثال:

- **القدرة على التأثير في متطلبات المكوّن:** حتى لو لم تكن المتطلبات

متطابقة 100 في المئة، فإن القدرة على التأثير في مسار المكوّن ستكون حاسمة.

● **مسار المكوّن:** قد يتضمن مسار المكوّن المعلن عناصر ليست ذات صلة بالمنتج حالياً، لكنها قد تكون مستقبلاً.

● **الشهرة:** قد يكون المكوّن قد أسس شهرة في ما يتعلق بخصائص الجودة المهمة.

● **الانفتاح:** عند الإعلان عن الصناديق السود(*)، تتطلب العديد من المكوّنات مستوى من الفهم لتصميم المكوّنات الداخلية التي تجعل منها صناديق بيْضاً بكفاءة. يمكن القول إن هذا سبب مهم لعدم نجاح COTSs. إضافة إلى ذلك، الانفتاح عامل مهم لنجاح المكوّنات مفتوحة المصدر في صناعة البرمجيات الحالية.

سيُستخدم المكوّن الناجح في العديد من المنتجات التي قد يكون لديها بعض القواسم المشتركة، بغضّ النظر عن كونها تعتمد على المكوّنات بطريقة أو بأخرى.

قد تكون متطلبات المكوّن محكومة بواسطة عدد من العوامل، منها:

● الحصول على تغذية راجعة عن التحسينات المحتملة من العملاء الحاليين الذين يستخدمون المكوّن. في حال وجود علاقات تجارية بين المنتج والمستهلك، فقد يكون هناك بعض المصطلحات التعاقدية (مثلاً، في سياق عقد الدعم).

● تحليل تسويقي لمتطلبات المنتج في المنتجات التي لا تستخدم المكوّن حالياً. إن تطوير المكوّن ليدعم هذه المتطلبات يمثل فرصة لنمو حصة المنتج في السوق.

● عوامل خارجية كالمعايير. قد تكون متطلبات المكوّن مرتكزة (جزئياً) على مواصفات معيارية أو واقعية. عند تطوير مثل هذه المواصفات، يمكن أن يصبح دعم المواصفة المطوّرة متطلباً.

(*) يستعمل تعبير الصندوق الأسود (Black Box) في حالة عدم معرفة النموذج الداخلي (Model) للنظام (system) وطريقة العمل الداخلية، على عكس الصندوق الأبيض أو الزجاجي (Glass Box) حيث نعرف كيف يتم تحويل المداخل إلى المخارج (المترجم).

- عوامل داخلية، كتحسين عوامل الجودة التي تعتبر في غاية الأهمية بالنسبة إلى مطوّري المكوّن كإمكانية الصيانة. ومن العوامل الأخرى مدى اهتمام المطوّرين الشخصي في اكتشاف بدائل للتصميم أو العثور على مجالات تحسين صغيرة.

نتج من هذه المنهجية منهجيةٌ أخرى ذات طبيعة «الأسفل ـ الأعلى»، حيث إنه بدلاً من أن تكون تابعة لمجموعة معيّنة من المنتجات (أي منهجية «الأعلى ـ الأسفل»)، هناك توجه لأن تكون العملية ذات مسار من الأسفل إلى الأعلى حيث يتم اختيار المكوّنات الأقل أو الأكثر استقلالية ويتم وضعها معاً لتحقق متطلبات المنتج. بهذه الطريقة، يتم حل التعارض المحتمل بين المكوّنات المطوّرة المستقلة عن طريق اختيار المكوّنات وعملية الدمج.

2 ـ 4 ـ 2 إدارة الجودة والهيكلية

من ميزات المنهجية التركيبية عدم وجود هيكلية مركزية. بدلاً من ذلك، هناك هيكلية خاصة بكل منتج وبكل شريحة هيكلية. وهذا يضع عدداً من التحديات أمام البحث في ما يتعلق بتطبيق أساليب تقييم الجودة، كمثال، وهذا يفترض وجود هيكلية مدارة مركزياً وسيطرة كاملة على الموجودات التي تحكمها الهيكلية.

إن الهدف من تطبيق أساليب تقييم الجودة هو التحقق من مطابقة متطلبات الجودة المدارة مركزياً (وهذا غير متوفر في المنهجية التركيبية) ولتحسين جودة المنتج من طريق تحديد القضايا المتعلقة بالجودة. على كلٍّ، إن تنفيذ تقييم الهيكلية على مستوى المنتج يجعل الأمر منطقياً نسبياً نظراً إلى غياب التحكم بالمكوّنات المشكّلة.

- نتيجة ذلك، يجب أن يتم تقييم الجودة والتحسينات عليها على مستوى المكوّن أو على مستوى الشريحة الهيكلية. على كلٍّ، في ظل غياب إدارة متطلبات الجودة المركزية وغياب السيطرة على المكوّنات المستقلة والتابعة، فإن ذلك يعني أن ضمان جودة متطلبات النظام كقيود الزمن الحقيقي والمخرجات والأمن وغيرها، من الصعوبة بمكان. يجب أن يتوقع مطوّرو المكوّنات متطلبات الجودة التي يتطلبها العملاء المحتملون ويحولونها إلى مطلب لتحسين المكوّن.

- من القضايا الأخرى، أن معظم أساليب تقييم الهيكلية تتطلب نظاماً متكاملاً. في المنهجية التركيبية، يُفضل أخذ تأثيرها في الجودة في الاعتبار قبل أن يتم دمج المكوّنات. قد تؤدي أي من المشكلات التي يتم تحديدها إلى تحسين في المكوّن، لكنها قد تؤدي أيضاً إلى اختيار مكوّنات بديلة.

الهدف الثاني من وجود هيكلية واضحة للبرمجية هو فرض نمط الهيكلية وقواعد التصميم. وسبب ذلك هو أن ذلك يضمن أن تكون مكوّنات الهيكلية مناسبة لبعضها بعضاً. ثمة مشكلة تتعلق بالمكوّنات الجاهزة للاستخدام التجارية وهي البحث عن مكوّنات ذات واجهات متطابقة. إن غياب الهيكلية المحكومة مركزياً لا يعني عدم وجود قاعدة موجّهة للهيكلية. لكن بالضرورة أن المكوّنات التي ستستخدم مع بعضها بعضاً يجب أن تشترك في هيكلية. على الأقل يتطلب ذلك وجود مستوى معيّن من التوافقية. هذا ويمكن تخطي الفروقات الصغيرة باستخدام الشيفرة البرمجية. لكن إن القيام بكميات معيّنة من الشيفرة البرمجية ليس بالأمر المحدد عند إنشاء المنتجات. بناء على ذلك، تطرح التوافقية عدداً من التحديات الجديدة:

- كيفية توثيق الخصائص الهيكلية للمكوّنات والشرائح الهيكلية.
- كيفية تحسين هيكلية المنتج على النحو الأمثل بحيث تكون الأمثل للمكوّنات التي ستتكون منها.
- كيفية تصميم المكوّنات بحيث لا تفرض العديد من القيود أيضاً.

2 ـ 4 ـ 3 تكنولوجيا المكوّنات البرمجية

تم تأييد البرمجة ذات التوجه المرتكز على المكوّنات وخدمات الويب اللاحقة باعتبارها خطوة مهمة نحو بناء نظم برمجية كبيرة[18]. وسبب ذلك بالدرجة الأولى هو توفير بنى تحتية معيارية للمكوّنات مع واجهة برمجة تطبيقات API معرّفة جيداً إضافة إلى الخدمات ودعم التوافقية. وعلى الرغم من أن ذلك تطور كبير، إلا أنه لم يوضح القضايا المتعلقة بالتغيير والهيكلية كما في خطوط المنتجات البرمجية.

في خطوط المنتجات البرمجية والعديد من الأدوات المستخدمة في إدارة التهيئة، يتم إدارة التبعيات بين المكوّنات. فالتبعيات المدارة هي مكوّنات أساسية للمنهجية التركيبية. لكن، يشمل ذلك إدارة تبعيات الشيفرة البرمجية

التركيبية فقط ـ مثلاً، في ما يخص النسخات المختلفة وتوافقية واجهة برمجة التطبيقات (Application Programming Interface - API) .

كما قد تتضمن معلومات هذه الأنظمة على تهيئات مختبرة، ويمكن بالتالي أن تستخدم لوصف الشرائح الهيكلية.

الأمر المشوق في هذا السياق هو العمل المعروض في المرجع[14] . فان أوميرينغ (Van Ommering) هو المؤيد والنصير لما يُسمى بتسكين خطوط المنتجات ويقدم تقنية المكوّنات التي تدعم تلك الفكرة. إن منهجيته شبيهة جداً بمنهجيتنا لكنها تركز على المظاهر التقنية للمكوّنات أكثر من التركيز على المظاهر الأخرى التي ناقشناها في هذا الفصل.

بالنسبة إلى الغرض من خطوط المنتجات البرمجية التركيبية، نحتاج إلى مكوّنات يمكن أن تعمل في سيناريوهات الاستخدام والبيئات غير المتوقعة والمفاجئة. وهذا يعني، مثلاً، إدراك وقوة للتبعيات والتفاعلات مع المكوّنات الأخرى.

وبشكل أكثر تحديداً، التحديان الرئيسيان اللذان ناقشناهما هنا هما التبعيات الدلالية التي يجب أن يتم إدارتها بطريقة أكثر وضوحاً، والثاني هو كون المكوّنات مصممة بطريقة أكثر قوة.

إن التفاعل الدلالي (Semantic Interaction) بين المكوّنات يعني أنه يجب التكيّف مع سلوك المكوّن عن تركيبه مع مكوّنات أخرى. وهذا يتعدى التوافق النحوي (Syntactic Compatibility) لواجهات برمجة التطبيقات وتم التحقق منه بتوسع في سياق البحث في تكامل المزايا[6] . هذا وقد تكون هذه التفاعلات إيجابية أوسلبية:

- يتطلب التفاعل الإيجابي إضافة وظائف إضافية. على سبيل المثال، إذا كان لدينا تطبيق للبريد الإلكتروني خاص بالهواتف المحمولة وتطبيق لعرض الصور، فيجب أن يكون بالإمكان عرض الصور الواردة عبر البريد الإلكتروني.

- يتطلب التفاعل السلبي منا رفض إجازة بعض الحالات أو حذف الالتباس والغموض. عادة ما يتناقض مكوّنان أو ميزتان في سلوكهما أو تتنافسان على الموارد المحدودة. مثلاً، وضع الهاتف النقال على وضعية السكون ينبغي ألاّ يعطل ساعة المنبه فيه.

في الواقع، تحدث بعض التبعيات عندما يتم جمع العديد من المكوّنات فقط ولا يمكن مراقبتها عندما يقتصر الأمر على ميزتين فقط في الوقت الواحد[16]. قد تكون المشكلات المتعلقة بالمزايا السلبية المرتبطة بالتكامل صعبة، ولا يمكن تحديدها بسهولة في منهجية اختبار التكامل غير المركزي الموضحة سابقاً.

2 ـ 4 ـ 4 العملية وقضايا التنظيم

يشير المرجع[4] إلى أن هناك العديد من الطرق المستخدمة لتنظيم خطوط المنتجات البرمجية. إن تبني المنهجية التركيبية يجعل من الممكن ومن الضروري التنظيم بأسلوب مختلف. فكما ناقشنا في أقسام سابقة، من الخصائص الأساسية للمنهجية التركيبية أن هناك إدارة أقل مركزية للمتطلبات والهيكلية والتنفيذ. أساسياً، يحدث التطوير والتطور للمكوّنات بطريقة غير مركزية. إن فصل تطوير المنتج عن تطوير المكوّن هو هدف صريح للمنهجية التركيبية لأنها تتيح اتخاذ القرارات التي تؤثر في المنتجات التي يجب أن تكون منفصلة عن اتخاذ قرارات تؤثر في المكوّنات.

إن القيام بذلك في الشركة نفسها يؤدي إلى تناقض في إدراك أن للشركة أهدافاً وتوجهات ورؤية. إن جميع الأنشطة التي تقوم بها الشركة (بما في ذلك تطوير المكوّنات والمنتجات) تتبع هذه الرؤية الشاملة (أو يجب أن تتبع انطلاقاً من تلك الرؤية). وهذا يعني أن تطوير المنتجات غير مستقل عن تطوير المكوّنات إطلاقاً. وبالتالي، فإن تقديم منهجية تركيبية في شركة تعمل على تطوير المنتجات ينطوي على عدد من التحديات:

- كيفية التنظيم، إذ إن فرق تطوير المنتج لها حرية اختيار المكوّنات الخارجية أو بدء تطوير مكوّنات جديدة بدلاً من استخدام المكوّنات المطوّرة داخلياً. إن قرارات الأعمال التي تتخذها فرق العمل بشأن عدم استخدام المكوّنات المطوّرة في الشركة نفسها له عواقب سلبية على فرق تطوير المكوّنات. فقد لا تكون أفضل الحلول التقنية هي الأفضل بالنسبة إلى الشركة كلها. إن موازنة هذه الأمور هو التحدي الرئيس الذي يجب تحديده.

- إن السماح لفرق تطوير المكوّنات بالاضطلاع بمسؤولية المخططات والهيكليات الخاصة بالمكوّنات خاصتهم من شأنه أن يؤدي إلى وضع يتم فيه إنفاق الموارد على تطوير المزايا والمنافع التي لن تستخدم من قبل أي فريق من فرق تطوير المنتج. إن موازنة ابتكارات وأفكار فرق تطوير المنتج وفرق تطوير المكوّنات أمر في غاية الأهمية للسيطرة على تكاليف التطوير كما هو مخطط له.

- من القضايا التي تبرز في أي شركة، عملية توزيع الموارد (المال والأشخاص والوقت وغيره) على وحدات العمل في الشركة. أساسياً، تتنافس فرق تطوير المنتج وفرق تطوير المكوّن على الموارد نفسها. لكن فرق تطوير المنتج هي وحدات العمل الوحيدة التي تساهم مباشرة في إيرادات الشركة، وهذا يقود إلى انحياز تفضيلي لها. لذا، فإن تبنّي منهجية تركيبية يقتضي إنشاء سلسلة القيمة الداخلية أو آلية تسويق لتوزيع الموارد الداخلية.

- في حين أنه قد تستخدم المكوّنات من قبل فرق المنتج فقط، فقد تصبح المكوّنات نفسها منتجات قد تهم شركات تطوير برمجيات أخرى. وحيث إن هذا لا يتعارض مع المفاضلة بين المنتجات، إلا أنه يفضل في تسويق مثل هذه المكوّنات بصورة منفصلة عن بعضها بعضاً، أو قد يتم مشاركة عبء تطويرها مع شركات أخرى حتى لو كانت شركات منافسة. إن تحويل المكوّنات إلى منتجات داخلياً أو الاعتماد على مكوّنات ذات مصادر مفتوحة هو من الآثار الجانبية الطبيعية لتطبيق المنهجية التركيبية، لكن قد يقود ذلك أيضاً إلى إيجاد متطلبات جديدة غير معرفة في نطاق تطوير المنتج الأصلي.

بالنسبة إلى هذه التحديات التنظيمية، يجب تحقيق توازن حذر بين الاهتمامات المتضاربة لفرق تطوير المنتج وفرق تطوير المكوّنات والمهمة العامة للعمل.

لكن، يجب ملاحظة أن هذا الأمر ينطبق على المنهجية غير التركيبية. إن سبب التحول من منهجية التطوير القائمة على خط المنتجات البرمجية إلى المنهجية التركيبية فهو أن الإدارة المركزية لهذه القرارات تصبح أصعب بنمو عملية التطوير في القياس، وأن نطاق خط المنتج البرمجي يصبح أوسع.

2 ـ 5 الملخص

أوجدت عائلات المنتجات البرمجية اعتماداً واسعاً على صناعة النظم المدمجة، إضافة إلى مجالات أخرى. ونظراً إلى نجاحها، تشهد عائلات المنتج في العديد من الشركات توسعاً في نطاق العائلة. لقد ناقشنا العديد من القضايا الأساسية التي يمكن أن تنشأ عن ذلك وعن أمور أخرى كالبرمجيات التجارية والبرمجيات الخارجية التي تكون خارج سيطرة المؤسسة.

بالنسبة إلى عائلات المنتج التي تهدف في المقام الأول إلى تغطية مدى واسع من المنتجات، فقد اقترحنا مفهوم عائلات المنتج التركيبية. تعتمد هذه المنهجية على تنظيم غير مركزي يعطي عملية إنشاء المنتج مرونة ومسؤولية أكبر. كما إنها تعطي حرية لمطوّري مكوّن المنتج الداخلي. بدلاً من دمج المنصة بالكامل، نعتمد على مفهوم شرائح الهيكلية الجديد لضمان التكامل والاختبار إلى ما بعد الاختبار على مستوى المكوّنات. إضافة إلى ذلك، بيّنا أن هذه المنهجية التركيبية يجب أن تتضمن جميع جوانب تطوير البرمجية لضمان نجاحها. كما أننا حددنا العديد من تحديات البحث الرئيسة التي تعترض المنهجية الجديدة في ما يخص المتطلبات وإدارة الجودة وتقنيات المكوّنات البرمجية إضافة إلى العمليات وتقسيم المسؤوليات في المؤسسة.

من الأمور التي تثير اهتمامنا في شركة نوكيا (Nokia)، حيث ارتكزت عملية التطوير على المنهجيات ذات التوجه التكاملي، أن السلوك المتشعب والمتطلبات غير الوظيفية كفعالية أداء الجهاز واستهلاك الطاقة. لقد أثبت ذلك صعوبة المنهجية ذات التوجه التكاملي البالغة التي نعتمد عليها حالياً.

المراجع

1. J.-J. Amor-Iglesias, J. M. González-Barahona, G. Robles-Mart'nez, and I. Herra'iz-Tabernero. Measuring Libre software using Debian 3.1 (Sarge) as a case study: Preliminary results. *UPGRADE: European Journal for the Informatics Professionals*: vol. 6, no. 3, 2005, pp. 13-16.

2. L. Bass [et al.]. Product Line Practice Workshop Report, Technical Report CMU/SEI-97-TR-003, Software Engineering Institute, June 1997.

3. J. Bosch. *Design and Use of Software Architectures: Adopting and Evolving a Product Line Approach.* Pearson Education (Reading, MA: Addison-Wesley & ACM Press), 2000.

4. J. Bosch. «Maturity and evolution in software product lines: Approaches, artefacts and organization.» paper presented at: *Proceedings of the Second Conference Software Product Line Conference (SPLC2)*, 2002, pp. 257-271.

5. J. Bosch. «Expanding the scope of software product families: Problems and alternative approaches.» Paper presented at: *Proceedings of the 2nd International Conference on Quality of Software Architectures (QoSA 2006)*, LNCS 4214, Springer, 2006, p. 1.

6. M. Calder [et al.]. «Feature Interaction: A Critical Review and Considered Forecast.» *Computer Networks*: vol. 41, no. 1, 2003, pp. 115-141.

7. D. Dikel [et al.]. «Applying software product-line architecture.» *IEEE Computer*: vol. 30, no. 8, 1997, pp. 49-55.

8. J. van Gurp. OSS Product Family Engineering. First International Workshop on Open Source Software and Product Lines at SPLC 2006. Available from <http://www.sei.cmu.edu/splc2006/>.

9. J. van Gurp, R. Smedinga, and J. Bosch. «Architectural design support for composition and superimposition.» paper presented at: *Proceedings of the 35th Hawaii International Conference on System Sciences (HICSS-35 2002)*, 2002, p. 287.

10. IEEE Std P1471-2000. *Recommended Practice for Architectural Description of Software- Intensive Systems.* New York: IEEE, 2000.

11. T. Kim [et al.]. «Dynamic software architecture slicing.» Paper presented at: *Proceedings of the 23rd International Computer Software and Applications Conference (COMPSAC '99)*, 1999, pp. 61-66.

12. R. R. Macala, L. D. Stuckey Jr., and D. C. Gross. «Managing domain-specific product-line development.» *IEEE Software*: vol. 13, no. 3, 1996, pp. 57-67.

13. R. van Ommering. «Building product populations with software components.» paper presented at: *Proceedings of the 24th International Conference on Software Engineering*, 2002, pp. 255-265.

14. R. van Ommering and J. Bosch. «Widening the scope of software product lines: From variation to composition.» paper presented at: *Proceedings of the 2nd Software Product Line Conference (SPLC2)*, 2002, pp. 328-347.

15. W. Pree and K. Koskimies. «Framelets: small and loosely coupled frameworks.» *ACM Computing Surveys*: vol. 32, no. 1, 2000, p. 6.

16. C. Prehofer. Feature-oriented programming: «A New Way of Object Composition.» *Concurrency and Computation*: vol. 13, no. 6, 2001, pp. 465-501.

17. E. S. Raymond. *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. Sebastopol, CA: O'Reilly & Associates, 1999.

18. M. Stal. «Web services: Beyond component-based computing.» *Communications of the ACM*: vol. 45, no. 10, 2002, pp. 71-76.

19. C. Szyperski. *Component Software: Beyond Object Oriented Programming*. Reading, MA: Addison- Wesley, 1997.

20. D. Wheeler. More than a Gigabuck: Estimating GNU/Linux's Size, <http://www.dwheeler.com/sloc/redhat71-v1/redhat71sloc.html>, 2002.

3

تعليم أنماط التصميم

بيرند بروغ (Bernd Brügge)
وتيمو وولف (Timo Wolf)

3 ـ 1 المقدمة

في البرمجة كائنية التوجه، تكون أنماط التصميم عبارة عن قوالب يمكن أن يعاد استخدامها مرات عديدة لحل طائفة من المشكلات متكررة الحدوث[2]. يتكون نموذج التصميم من الاسم ووصف المشكلة والحل والنتائج. تنتمي أنماط التصميم إلى مهندسي البرمجيات المختصين بالمعرفة الأساسية ومصممي الهيكلية ومطوّري النظم كائنية التوجه وتوفر لغة مشتركة بين المساهمين في مشروع تطوير برمجي.

يتضمن استخدام أنماط التصميم ما يأتي:

- معرفة بطائفة واسعة من أنماط التصميم.
- تحليل المشكلات وتحديد أنماط التصميم الممكنة.
- تطبيق أنماط التصميم في شيفرة البرمجة التصميمية.
- تحديد أنماط التصميم في شيفرة البرمجة التصميمية.

عندما كنا نُدرّس هندسة البرمجيات، بما في ذلك أنماط التصميم، أدركنا أن الطلاب كانوا يعانون مشكلة فهم أنماط التصميم وتطبيقها. فقد اكتفى بعض الطلاب بالنظر إلى نماذج نوع نمط التصميم المتوفرة في الفهارس المصورة، لكنهم لم يكونوا بالقدر الذي يسمح لهم ليستوعبوا مفاهيم تلك الأنماط

وسلوكها الديناميكي. لقد واجهوا صعوبات في تطبيق المعرفة لحل مسألة مُعطاة، من حيث فصل مفاهيم نمط التصميم وتطبيقاتها، وتحديد مفاهيم نمط التصميم المطبّق من شيفرة البرمجة المصدرية. على سبيل المثال، لاحظنا أن الطلاب يواجهون مشكلة في شرح النمط، وليس في شرح آليات Java Mouse Event Listener. لم يكن هؤلاء الطلاب قادرين على رؤية المفهوم نفسه.

لقد أدركنا أن الخبرة العملية مطلوبة لفهم واستيعاب تطبيق نمط التصميم. وعليه فقد طورنا مجموعة من التمارين على نمط التصميم مشروحة في هذا الفصل. ترتكز هذه التمارين على تطبيقاتنا في لعبة الكويكبات (Asteroids) القديمة، حيث تبني الهدف لتطبيق نمط التصميم. نغطي هنا مفاهيم نمط التصميم عن طريق تصاميم النمذجة وتطبيق الأنماط في لغة Java. لقد قمنا بتصميم التمارين لتكون أصغر حجماً وأكثر سهولة بحيث يستطيع الطلاب المبتدئون فهمها وإدراك ماهية التمارين في وقت قصير. إن النظام كبير بما فيه الكفاية بحيث تصبح مزايا تطبيق أنماط التصميم أكثر وضوحاً ولا تؤدي إلى المزيد من الأعباء التي قد تترتب عن تغيير النظام من دون الاعتماد على نمط تصميم.

تبدأ التمارين بنظام قيد العمل وتصميمه الأولي. يشكّل كل تمرين من هذه التمارين متطلباً يجب أن يتم حله قبل تطبيق نمط التصميم، بما في ذلك النمذجة وتطبيقها. بمقارنة التمارين في المرجع[4]، نجد أن جميع التمارين في هذا الفصل تزايُديَّة وتمتد لتشمل النظام قيد التطوير. توفّر كلُّ عيّنة نموذجاً كائناً مرتكزاً على النمط، إضافة إلى تطبيقه، وبناء الأساس للتمرين الذي يليه.

يوضح القسم 3 ـ 2 تصميم النظام العيّنة الذي نقوم بعرضه. ثمة درس عن تجميع وتنفيذ النظام في القسم 3 ـ 3. أما تمارين نمط التصميم، فقد تم توضيحها في الأقسام من 3 ـ 4 إلى 3 ـ 9 وقد شملت المراقب والموائم ونمط الاستراتيجية. ونختم بالاستنتاج، حيث نوضح خبراتنا في تنفيذ الدرس في القسم 3 ـ 10.

3 ـ 2 تصميم لعبة الكويكبات

في هذا القسم، نوضح لعبة الكويكبات (Asteroids) ونعرض التصميم الأساسي للنظام. جميع التمارين التالية تعتمد على هذا التصميم.

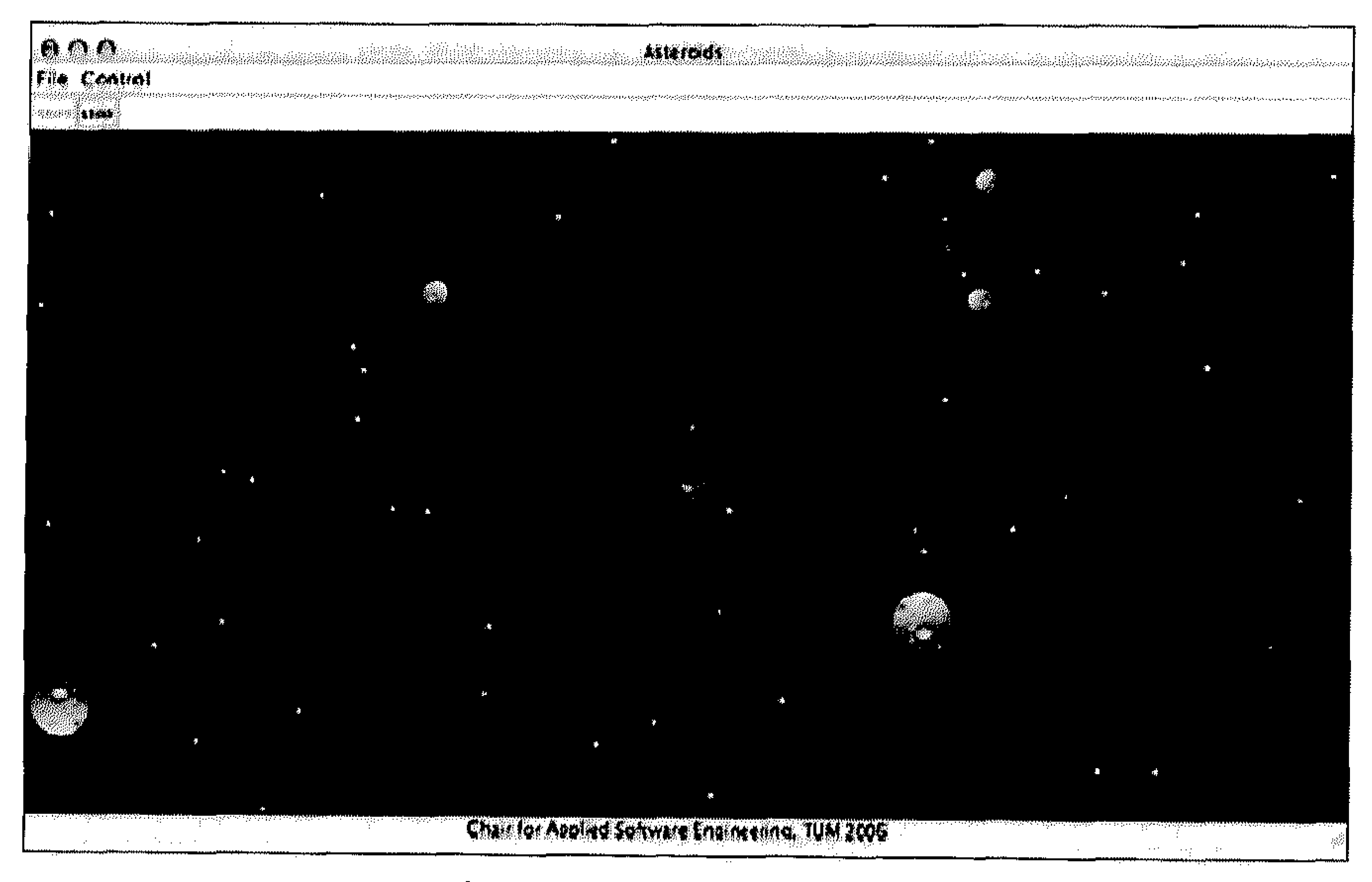
3 ـ 2 ـ 1 وصف اللعبة

يتحكم اللاعب بالمركبة الفضائية على لوحة اللعب التي تمثل الفضاء الخارجي. هناك نوعان من الكويكبات التي تظهر على لوحة اللعب أيضاً:

(أ) كويكبات صغيرة وسريعة، وهذه تغير اتجاهها فقط عندما تصطدم بأطراف لوحة اللعب.

(ب) كويكبات كبيرة وبطيئة، وهذه تغير اتجاهها دائماً.

قد تغيّر جميع الكوكيبات من سرعتها التي تتراوح من سرعة ضعيفة إلى السرعة القصوى. بإمكان اللاعب إطلاق الصواريخ وتغيير اتجاه وسرعة المركبة باستخدام لوحة المفاتيح. أما إذا ضُرب كويكب كبير بصاروخ فإنه يُستبدل بثلاثة كويكبات كبيرة وثلاثة كويكبات صغيرة. أما إذا ضُرب كويكب صغير، فإنه يختفي. يعتبر اللاعب فائزاً إذا دمّر جميع الكوكيبات. أما إذا اصطدمت المركبة الفضائية بأي كويكب فيُعتبر اللاعب خاسراً. يبيّن الشكل 3-1 لقطة للنسخة الأولية من لعبة الكويكبات.



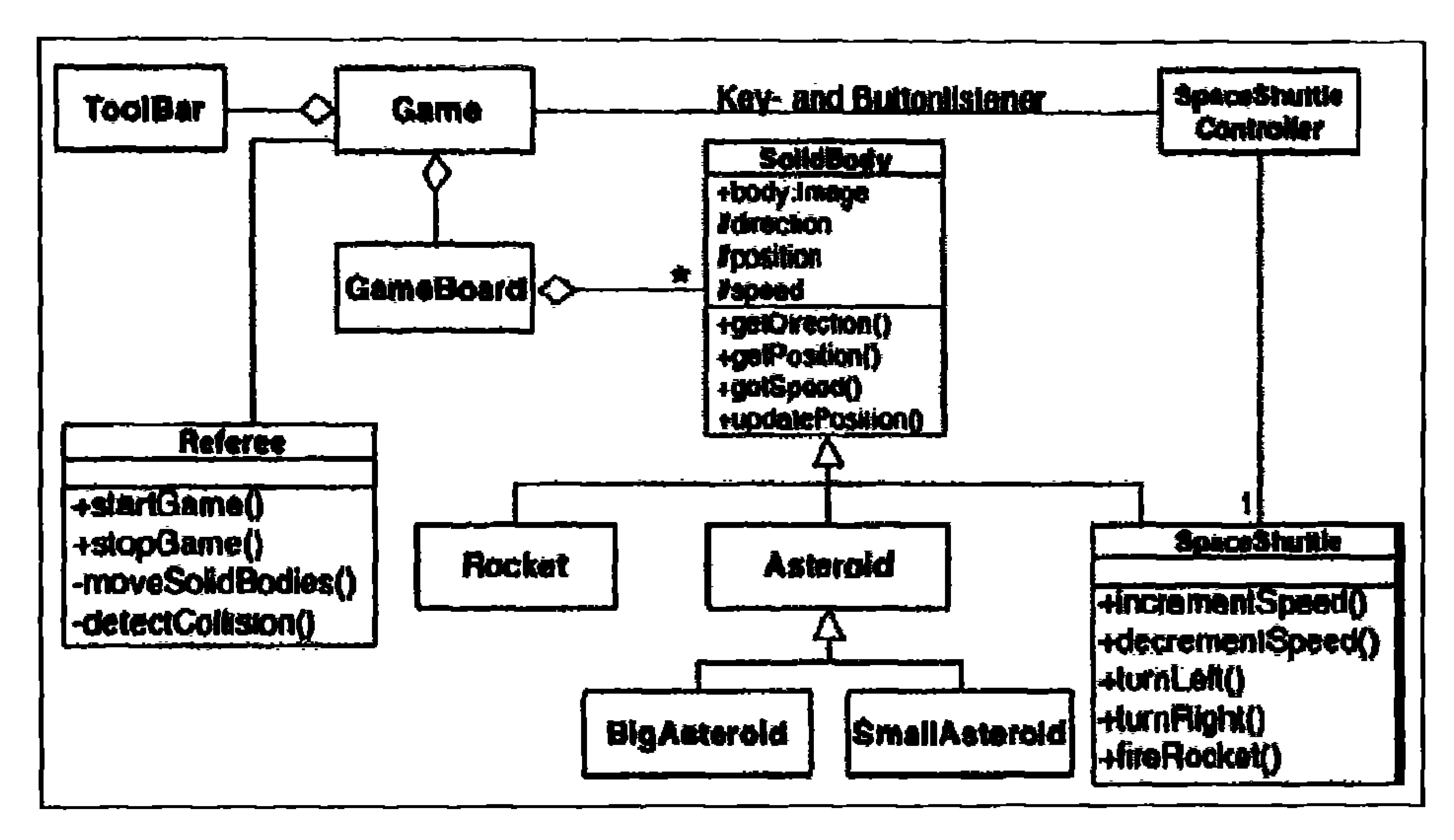
الشكل (3 ـ 1): لقطة للعبة الكويكبات (Asteroid) الأولية

بناءً على وصف اللعبة، وفّرنا تصميماً أولياً في الشكل 3-2. نركز هنا على الأنواع الأساسية والمفاهيم، وأهملنا التفاصيل غير الضرورية.

في ما يأتي نصِفُ الأنواع المبينة في الشكل 3-2.

3 ـ 2 ـ 2 النوع: Game

يمثل النوع الرئيس المسمى Game الشاشة الرئيسة لبرنامج الكويكبات. يتكون من نوعين فرعيين: GameBoard وToolBar.



الشكل (3 ـ 2): التصميم الأولي للعبة الكويكبات (مخطط نوع بلغة النمذجة الموحدة)

3 ـ 2 ـ 3 النوع: ToolBar

هذا النوع هو عبارة عن مكوّن تصويري ويمثل أزرار البدء والتوقف التي سيستخدمها اللاعب.

3 ـ 2 ـ 4 النوع: SolidBody

يمثل هذا النوع حالة تجريدية لجميع الكوائن التي تتحرك فوق لوحة اللعبة. لهذا النوع SolidBody صفة تمثيل الصورة المستخدمة للتلوين. يتم تخزين اتجاه الطيران في صفة الاتجاه. القيم المتاحة تتراوح من 0 إلى 360 وتمثل الاتجاه بدرجة الزاوية. القيمة صفر تمثل الاتجاه عندما يكون مشيراً إلى الأعلى. صفة الموقع لها إحداثيان عمودي وأفقي وتخزن موقع SolidBody الحالي. كل SolidBody له صفة السرعة. جميع الصفات محمية ضد التعديل من

قبل الكوائن الغريبة. كما تتوفر عمليات المُجلبات (Getter Operations) للوصول إلى صفات النوع SolidBody. تعمل العملية updatePosition () على حساب الموقع الجديد وتثبيته في الصفة اعتماداً على قيم صفات السرعة والاتجاه. إذا وصل الموقع الجديد إلى حدود اللوحة، يتم تغيير اتجاه الجسم الصلب حسب القاعدة: زاوية السقوط تساوي زاوية الانعكاس.

3 ـ 2 ـ 5 النوع: Asteroid

يمثل الكويكب الذي يتحرك على لوحة اللعبة. هذا النوع يستبدل قيمة عملية updatePosition () ويعمل على زيادة أو تقليل السرعة بشكل عشوائي.

3 ـ 2 ـ 6 النوع: BigAsteroid

يخصص النوع BigAsteroid النوع Asteroid عن طريق إضافة وظيفة إضافية تعمل على تغيير الاتجاه عشوائياً.

3 ـ 2 ـ 7 النوع: SmallAsteroid

يمثل الكويكب الصغير الذي يتحرك بسرعة أكبر من سرعة الأجسام في النوع BigAsteroid.

3 ـ 2 ـ 8 النوع: SpaceShuttle

المركبة الفضائية هي جسم صلب يتحكم به اللاعب. يخسر اللاعب اللعبة إذا دمرت المركبة الفضائية. ويفوز إذا كانت المركبة الفضائية هي الجسم الصلب الوحيد الذي يتبقى على اللوحة. يوفّر النوع عمليات عامة لتغيير السرعة والاتجاه وإطلاق الصواريخ.

3 ـ 2 ـ 9 النوع: Rocket

يتم إنشاء الأجسام الصاروخية بواسطة العملية fireRocket () التابعة للنوع SpaceShuttle. يكون اتجاه الصاروخ بالاتجاه نفسه للمركبة الفضائية وهو لا يتغير. تعمل الصواريخ على تدمير الأجسام الصلبة الأخرى. يتم حذف الصواريخ من اللوحة عندما تصل إلى حدود GameBoard أو عندما تصطدم بأجسام صلبة أخرى.

3 ـ 2 ـ 10 النوع: GameBoard

يمثل هذا النوع الفضاء الخارجي للكويكبات. يتكون من العديد من الأجسام. في أثناء اللعب، تتحرك الأجسام الصلبة ضمن GameBoard. يرسم GameBoard الصور للنوع SolidBodies في مواقعها. أما مواقع الأجسام الصلبة فتكون دائماً ضمن سطح اللوحة.

3 ـ 2 ـ 11 النوع: Referee

هذا النوع ـ Referee هو المتحكم الرئيس في الكويكبات. وهو مسؤول عن بدء اللعبة والتحكم بها وإنهائها. عند بدء اللعبة، يتم إنشاء عملية Thread منفصل للتحكم يقوم باستدعاء العملية moveSolidBodies () في فترات متكررة. هذه العملية moveSolidBodies () تستدعي عملية أخرى updatePosition () لجميع الأجسام الصلبة وتجبر GameBoard على إعادة رسمها ما يمكن SolidBoard من التحرك. بعد تحريك جميع الأجسام الصلبة، تُستخدم العملية detectCollision (...): SolidBody لتحديد التصادمات. فهي تقوم بحساب إذا ما تقاطع جسمان body1 وbody2 وتقرر أي الجسمين يتم تدميره. والجسم الصلب الذي تم تدميره يُحذف. إذا تقاطع كويكبان، لا يتم تدمير أي منهما. إذا تقاطع كويكب أو المركبة الفضائية مع صاروخ، يتم تدميرهما. إذا تقاطع كويكب مع مركبة الفضاء، يتم تدمير المركبة. بعد تحديد التصادمات، باختبار النوع Referee إذا ربح اللاعب أم خسر اللعبة. يفوز اللاعب إذا لم يتبقَ أي كويكب على اللوحة، ويخسر إذا دمرت المركبة الفضائية. في هذه الحالات، أو إذا استدعيت العملية stopGame () يتم إيقاف عملية التحكم واللعبة.

3 ـ 2 ـ 12 النوع: SpaceShuttleController

يخضع هذا النوع إلى حركات لوحة المفاتيح والفأرة. ويتم استدعاء العمليات الخاصة بالمركبة الفضائية حسب هذه الحركات لتغيير الاتجاة والسرعة أو لإطلاق صواريخ جديدة.

3 ـ 3 تنزيل لعبة الكويكبات وتشغيلها

يوضح هذا القسم كيفية تنزيل لعبة الكويكبات وتجميعها وتنفيذها. تم تطوير هذه اللعبة من قبل رئيس قسم هندسة البرمجيات التطبيقية في جامعة

ميونيخ التقنية (TUM). جميع المصادر متاحة للاستخدام من خلال بوابة مشروع لعبة الكويكبات ذي كلمة المرور المحمية. يمكن طلب عنوان الصفحة الإلكترونية وكلمة المرور واستلامهما من خلال البريد الإلكتروني. في هذا الفصل، نستخدم المتغير $Asteroids كمخزن لهذا العنوان. توفّر البوابة الإلكترونية بوابات فرعية لجميع التمارين والحلول. سنقوم بشرح عملية تجميع وتنفيذ النسخة الأولية للعبة الكويكبات الموضحة في القسم 3-2.

قم بتنزيل الشيفرة المصدرية للعبة من الموقع: <http://$Asteroids/ 1_InitialGame/downloads> ثم قُم بفك ملف الأرشفة. ستجد الملفات والفهارس الآتية:

- الفهرس src: يتضمن جميع ملفات الشيفرة المصدرية بلغة جافا، إضافة إلى ملفات الصور والصوت الخاصة باللعبة.
- الفهرس etc: يتضمن جميع ملفات Scripts اللازمة لبدء اللعبة من خلال منصات إطلاق مختلفة كـ Windows Linux, Mac OS X.
- الملف build.xml: هذا هو ملف Ant المسؤول عن تجميع اللعبة.

لتجميع وتشغيل اللعبة، يتطلب الأمر وجودJava 1.5 وApache-Ant في الحاسوب. يمكنك تخطي الخطوات الآتية إذا كان كلٌ من Ant وJava SDK مثبتان في الجهاز.

- قُم بتنزيل (Java SDK (Download) الإصدار 1.5 أو إصدار أعلى.
- يمكنك تنزيل Java J2SE 5.0 من الموقع التالي: <http://java.sun. com/j2se/1.5.0/download.jsp>
- ثبّت جافا واتبع تعليمات التثبيت.
- ملاحظة: قد تتطلب بعض التطبيقات وجود المتغير JAVA_HOME في بيئة النظام. تأكد من وجود هذا المتغير، ومن أنه معدّ إعداداً صحيحاً.
- قُم بتنزيل (Download) آخر إصدار من Apache-Ant من الموقع: http://ant.apache.org/bindownload.cgi
- ثبّت (Install) Ant باتباع التعليمات الواردة في دليل الاستخدام الخاص بـ Apache Ant: http://ant.apache.org/manual/index.html

بعد تثبيت جافا وAnt بنجاح، يمكننا تجميع لعبة الكويكبات وتشغيلها. افتح سطر كتابة الأوامر، ثم افتح الفهرس الذي عملت على فكّه مسبقاً. أما في شاشة (Windows)، فيبدأ تنفيذ الأوامر من سطر الأوامر (Command Line Shell)، بينما في النظم التي تعمل من خلال يونيكس (Unix-based)، يمكن استخدام أي أداة ك Bash. توضح الخطوات التالية مهام Ant والتي أخذت من الملف build.xml الذي تم تحميله:

- اطبع أمر التجميع في الأداة ant compile لتجميع شيفرة البرمجة الخاصة باللعبة. يتم تجميع ملفات الأنواع المكتوبة بلغة جافا في فهرس أنواع جديد. يتم نسخ جميع ملفات الصور والصوت من الفهرس src إلى فهرس classes.

- إن طباعة الأمر ant يستدعي الهدف التلقائي المسمى build الذي ينشيء الملفات التنفيذية لجميع منصات النظام المدعومة. سيتم إنشاء فهرس جديد يتضمن فهرسين فرعيين هما Asteroids وOSX. يتكون الفهرس Asteroids من الملف Java Archive asteroids.Jar الذي يحتوي على جميع أنواع جافا المجمعة، إضافة إلى Startup Scripts لتشغيل اللعبة ضمن نظامَي Windows وUnix. أما الفهرس OSX فيحتوي على الملفات التشغيلية اللازمة لتشغيل اللعبة ضمن نظام Mac OS X.

- اطبع الأمر ant clean لحذف الأنواع والفهارس. يمكننا تنفيذ لعبة الكويكبات، بعد تجميع الملفات وبنائها. يتم تجميع أنواع جافا المجمّعة والمصادر ومعلومات البدء المتطلبة في ملف Java archive build/Asteroids/asteroids.jar. يتم تشغيل اللعبة عن طريق النقر بواسطة الفأرة نقرة مزدوجة على ملف asteroids.jar وينطبق ذلك على جميع منصات الإطلاق. إضافة إلى ذلك، تتوفر ملفات البدء لما يأتي:

- Windows: افتح المتصفح، ثم انتقل إلى الفهرس build/Asteroids. أنقر مرتين على الملف asteroids.bat.

- النظام المبني على Unix: افتح shell، ثم انتقل إلى الفهرس build/Asteroids. لبدء اللعبة، اطبع «./asteroids.sh».

- Mac OS X: افتح Finder ثم انتقل إلى الفهرس build/OSX. انقر مرتين على تطبيق الكويكبات للبدء.

3 ـ 4 التمرين الأول: نمذجة نمط المشاهد

في التمرين الأول، يجب أن يتم تغيير التصميم الأولي (انظر الشكل 3 ـ 2) حتى يتوافق مع المتطلبات الآتية:

يجب أن يرى اللاعب المعلومات المناسبة عن المركبة الفضائية في أثناء اللعب. يجب أن تتضمن هذه المعلومات سرعة المركبة الحالية واتجاهها وموقعها. حتى يتم إدراك المتطلبات، يجب أن تضاف لوحة للقياسات إلى نموذج التصميم الأولي للعبة. تتكوّن لوحة القياسات من الأدوات التي تعرض حالات المركبة كالسرعة والاتجاه والموقع. يجب أن يشتمل التصميم على الأدوات الجديدة، وبذا يتطلب تقارناً أقل للحد من أعباء التغيير التي قد تطرأ مستقبلاً. الأدوات المتطلبة لهذا التمرين هي:

- عداد السرعة ويعرض سرعة المركبة.
- بوصلة، وتعرض اتجاه المركبة.
- نظام تحديد المواقع GPS، ويعرض موقع المركبة.

يجب أن تعرض هذه الأدوات التغيرات التي تطرأ على حالة المركبة الفضائية باستمرار. استخدم Observer Pattern برويدج (B. Bruegge and A. H. Dutoit, 2003[1]، ص 702) لعرض تغييرات الحالة بواسطة الأدوات.

مهمة التمرين

- ارسم مخطَّطَ نوعٍ بلغة النمذجة الموحّدة، يشتمل على المتطلبات المعطاة.
- ارسم مخطَّطَ تتابعٍ بلغة النمذجة الموحدة، يبيّن التفاعلات بين المركبة الفضائية والأدواتَ.

يجب أن يركّز نموذج التمرين على مفاهيم Observer Pattern ويجب أن يتضمن الصفات والعمليات اللازمة لفهم التصميم فقط.

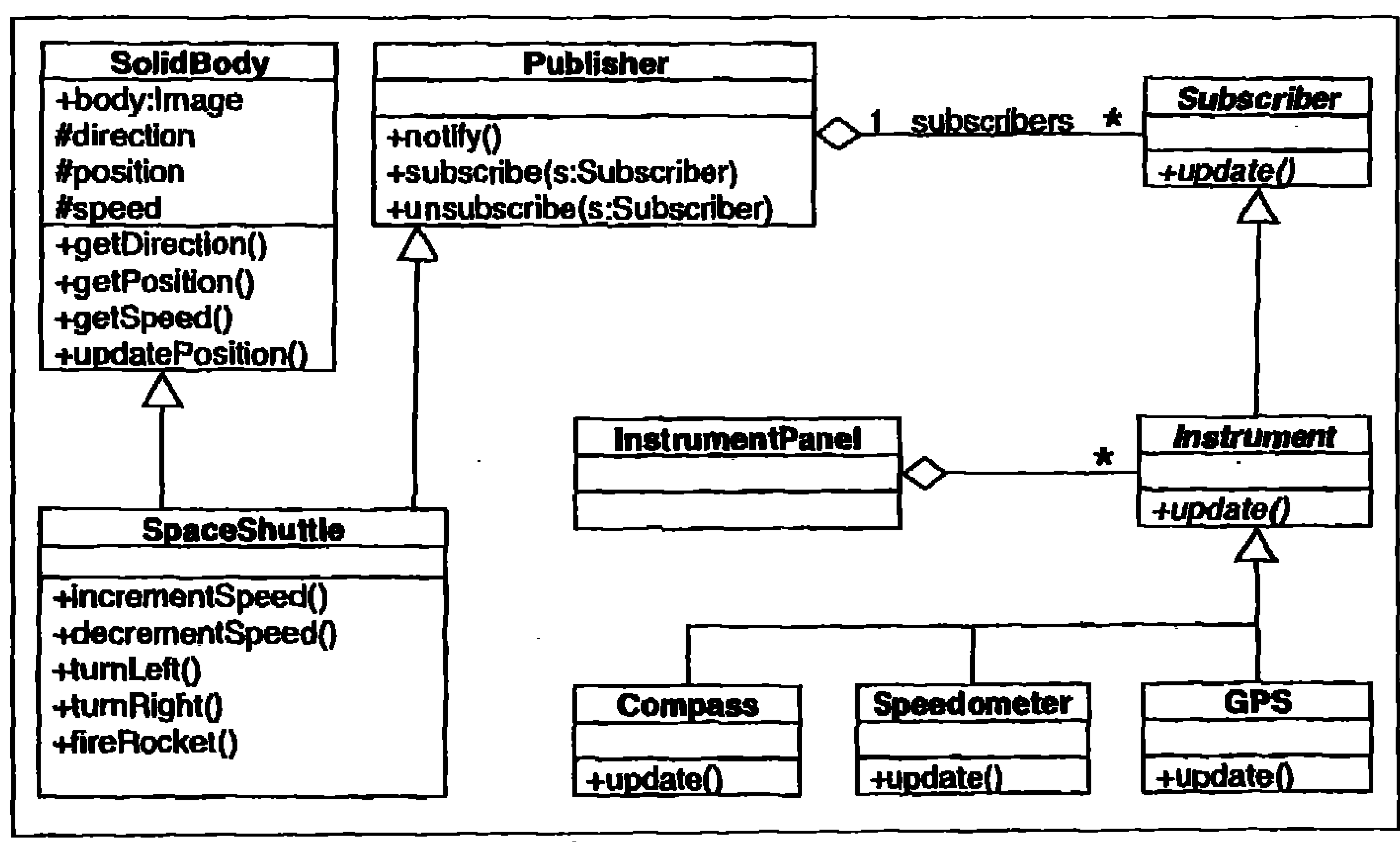
هذا ويمكن اهمال التفاصيل المتعلقة بالأدوات كالبوصلة ونظام تحديد المواقع والأنواع الموجودة أصلاً غير المرتبطة بالحل.

3 ـ 4 ـ 1 نموذج حل التمرين الأول

يبيّن مخطط النوع بلغة النمذجة الموحدة في الشكل 3-3 نموذجاً لحل

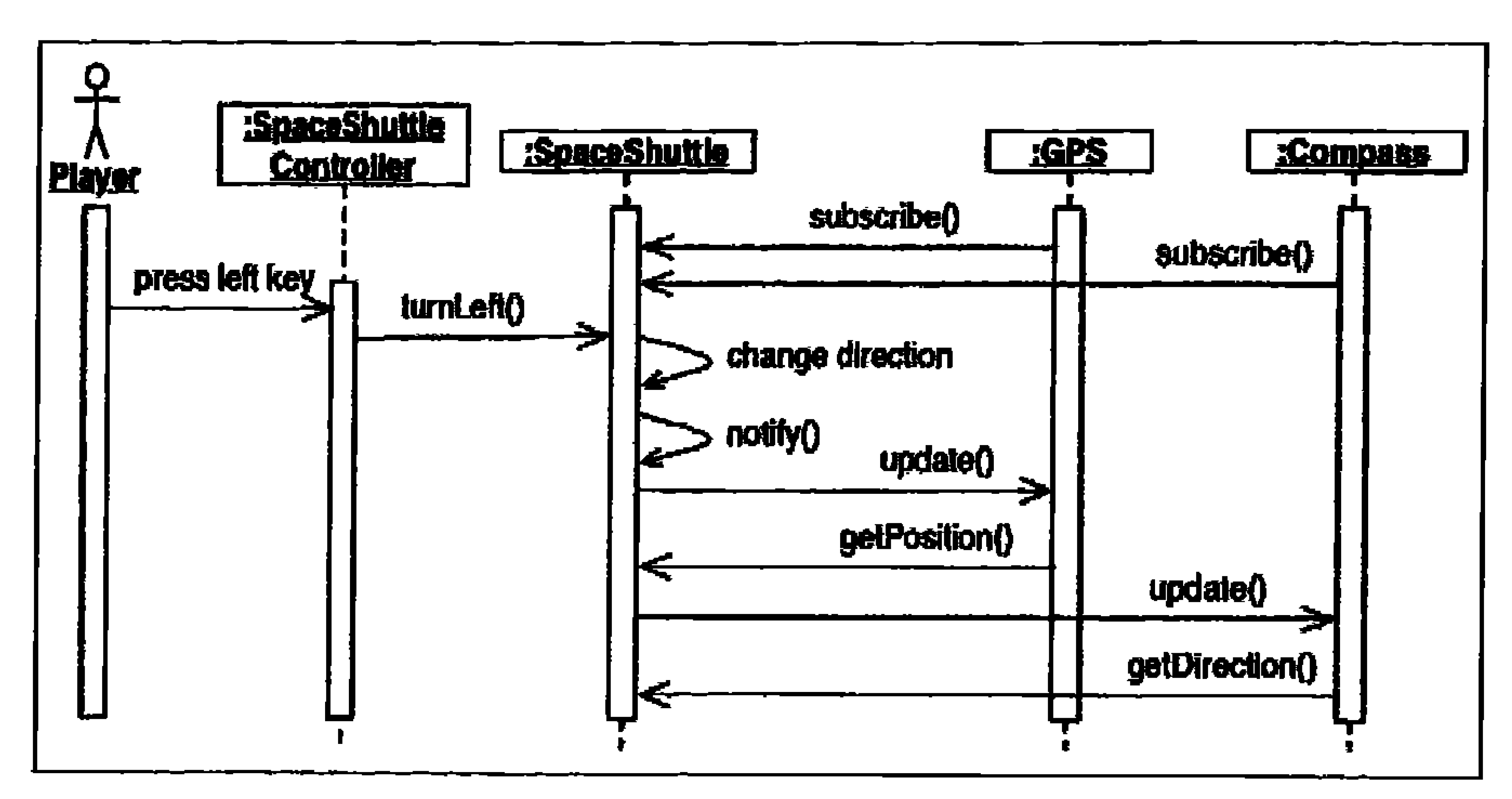
التمرين الأول. تم عرض الأنواع ذات العلاقة فقط مع التركيز على Observer Pattern والأدوات.

أولاً، قمنا بإضافة أنواع نطاق التطبيق الموضحة في التمرين. قمنا بإنشاء نوع جديد باسم InstrumentPanel ويتكون من أدوات عديدة. أما النوع Instruments فهو نظري فحسب ويوضح عموميات عن الأنواع الخاصة بالأدوات الأساسية كالبوصلة وعدّاد السرعة ونظام تحديد المواقع. يقلل النوع Instruments من كمية الارتباطات بين InstrumentPanel والأدوات الملموسة. وبخلاف ذلك، سيكون للنوع InstrumentPanel ارتباط مع كل أداة من الأدوات الملموسة، كما سيتطلب الأمر إنشاء ارتباطات جديدة عند إضافة أدوات جديدة. لقد طبقنا Observer Pattern لإشعار الأدوات بالتغيرات التي تطرأ على المركبة الفضائية. في هذه الحالة، يصبح النوع SpaceShuttle بمثابة الناشر إذ إنه يوفر المعلومات عن الوضع الحالي، بينما يصبح النوع Instrument بمثابة المشترك. والمشترك حالة يمكن من خلالها الاشتراك أو عدم الاشتراك مع الناشر، وبالتالي مع النوع SpaceShuttle. إن عملية الإشعار notify () الخاصة بالناشر تستدعي عملية التحديث update () لجميع الأدوات المشترِكة لإعلان تغيرات حالة المركبة. تنفذ جميع الأدوات العملية update (). فهي تسترجع المعلومات المطلوبة من SpaceShuttle وتحدّث المعلومات التي تعرضها عندما يتم استدعاء العملية update ().



الشكل (3 ـ 3): تصميم الكوائن الذي يتضمن الأدوات و Observer Pattern

يعرض الشكل 3 ـ 4 التفاعل بين الأدوات ممثلاً بمخطط تسلسل بلغة النمذجة الموحدة. هذا التصميم قابل للتوسع، إذ لا يُلزم إجراء أي تغيير على الأنــواع SpaceShuttle وInstrumentPanel وInstrument عــنــد إضــافــة أدوات جديدة. لاحظ أن التصميم يستخدم العديد من حالات التورّث للنوع SpaceShuttle وهذه الخاصية غير مدعومة في بعض لغات البرمجة. لقد وفّرنا تصميماً ذا تنفيذ مستقل يوضح المفهوم. هذا ويمكن إدراك المفهوم باستخدام أي لغة برمجة. يجب أن لا يعكس التصميم تركيب تصميم النوع بالضبط.



الشكل (3 ـ 4): التفاعل بين المركبة الفضائية وأدواتها

3 ـ 5: التمرين الثاني: برمجة Observer Pattern

يركّز التمرين الثاني على تطبيق التصميم الكائني الذي يتضمن Observer Pattern من التمرين الأول. يعمل التمرين على شيفرة تحويل العوامل المصدرية التي تنفّذ أجزاء التصميم. قُم بتنزيل شيفرة التحويل المصدري من الموقع: <http://$Asteroids/2_ObserverPatternExercise/downloads> .

قُم بتجميع وتنفيذ اللعبة كما هو موضح في القسم 3-3. هذا وستجد أن عدّاد السرعة ونظام تحديد المواقع وObserver Pattern منفذة من قبل بناء على التصميم الكائني من التمرين الأول.

مهمة التمرين:

نفّذ النوع Compass مبيناً اتجاه المركبة. يجب أن ينشر النوع Compass النوع

Instrument كما يجب أن يتم إضافته إلى InstrumentPanel. اعمل على اشتراك النوع Compass في المركبة الفضائية لاستلام إشعارات عن التغيرات التي تطرأ على الاتجاه من المركبة. قبل البدء في التنفيذ، نرغب في شرح الأنواع الأساسية في هذا التمرين. يستخدم التصميم الوارد في التمرين الأول عدداً من حالات التورّث للنوع SpaceShuttle. لغة الجافا لا تدعم حالات التورّث العديدة. وبناء على ذلك، قمنا بدمج النوع Publisher مع النوع SpaceShuttle عن طريق إضافة الـ methods المطلوبة.

```
public class SpaceShuttle extends SolidBody {
private ArrayList < SpaceShuttleSubscriber >  subscribers;
:
:
public void incrementSpeed () {...}
public void decrementSpeed () {...}
public void turnRight () {...}
public void turnLeft () {...}
public void fireRocket () {...}
:
:
protected void setSpeed (int speed) {
super.setSpeed (speed);
notifySpaceShuttleSubscribers ();
}
protected void setDirection (int direction) {
super.setDirection (direction);
notifySpaceShuttleSubscribers ();
}
public void subscribe (SpaceShuttleSubscriber subscriber)
{...}
public void unsubscribe (SpaceShuttleSubscriber subscriber)
{...}
public synchronized void notifySpaceShuttleSubscribers ()
{...}
}
```

لاحظ الفروقات بين التصميم والتطبيق. في التصميم، كنا قد أضفنا النوع Subscriber لبيان أننا نستخدم Observer Pattern، حيث يمكننا أن نميّز هذا النوع من خلال واجهة الجافا التي تعرف ب SpaceShuttleSubscriber. يشير الاسم إلى أن المشتركين يمكنهم الاشتراك في المركبة الفضائية وزيادة القدرة على قراءة شيفرة البرمجة. يتعرّف النوع SpaceShuttle على الواجهة SpaceShuttleSubscriber التي قد تكون عبارة عن أي نوع تنفيذ. لا يحتاج النوع SpaceShuttle إلى أي تعديلات إذا أضيفت أنواع SpaceShuttle Subscriber جديدة.

```
ublic interface SpaceShuttleSubscriber {
void update ();
}
```

يعمل النوع Instrument النظري المكتوب بلغة جافا على تنفيذ SpaceShuttle Subscriber وينشر النوع JPanel المستخدم لتخزين الرسوم خاص بـ Swing. يعمل النوع Instrument متغير محمي يمكن أن تستخدمه الأنواع الفرعية لاسترجاع معلومات المركبة الفضائية وعرضها. بإمكان الأنواع الفرعية كعداد السرعة أن تضيف مكوّنات Swing رسومية لعرض معلومات المركبة. يتم استدعاء العملية update () عندما تتغير صفات (خصائص) النوع SpaceShuttle.

```
public abstract class Instrument extends JPanel implements
SpaceShuttleSubscriber {
protected SpaceShuttle spaceshuttle;
public Instrument (SpaceShuttle spaceshuttle) {
this.spaceshuttle = spaceshuttle;
}
public abstract void update ();
}
```

ثمة نوع آخر يستخدم لتخزين Swing الرسومية وهو InstrumentPanel، ويتضمن جميع الأدوات. يعمل هذا النوع على إنشاء حالات الأدوات ويشاركها

في SpaceShuttle . الجزئية الآتية تتضمن شيفرة البرمجة التي تبين تهيئة عداد السرعة ونظام تحديد المواقع:

```
public class InstrumentPanel extends JToolBar {
:
:
public InstrumentPanel (SpaceShuttle theSpaceShuttle) {
super (JToolBar.VERTICAL);
setFloatable (false);
this.spaceshuttle = theSpaceShuttle;
:
:
speedometer = new Speedometer (theSpaceShuttle);
theSpaceShuttle.subscribe (speedometer);
add (speedometer);
gps = new GPS (theSpaceShuttle);
theSpaceShuttle.subscribe (gps);
add (gps);
:
:
}
:
:
}
```

3 ـ 5 ـ 1 نموذج حل التمرين الثاني

في ما يأتي نعرض الأجزاء الرئيسة من حل التمرين الثاني، بيد أنك تستطيع تنزيل الشيفرة البرمجية كاملة من الموقع: /http://$Asteroids> 2_ObserverPatternSolution/downloads<

أولاً، نُنشىء النوع Compass المتطلب الذي يوسع النوع النظري Instrument. نضيف النوع JLabel لمستوعب النوع Instrument التي تعرض اتجاه المركبة الفضائية كنص. نقوم بتعديل النص كلما تم استدعاء العملية () update كلما تغير الاتجاه.

```
public class Compass extends Instrument {
private JLabel theLabel = new JLabel ("", JLabel.CENTER);
public Compass (SpaceShuttle spaceshuttle) {
super (spaceshuttle);
setLayout (new BorderLayout ());
add (theLabel, BorderLayout.CENTER);
theLabel.setText (getText (spaceshuttle.getDirection ()));
}
public void update () {
String newText = getText (spaceshuttle.getDirection ());
if (!newText.equals (theLabel.getText ())) {
theLabel.setText (newText);
}
}
private String getText (int direction) {
return "Direction: " + direction;
}
}
```

ثانياً، نُنشيء حالة جديدة ونشركها في SpaceShuttle ونضيفها إلى InstrumentPanel.

```
public class InstrumentPanel extends JToolBar {
:
public InstrumentPanel (SpaceShuttle theSpaceShuttle) {
super (JToolBar.VERTICAL);
:
compass = new Compass (theSpaceShuttle);
theSpaceShuttle.subscribe (compass);
add (compass);
:
}
:
}
```

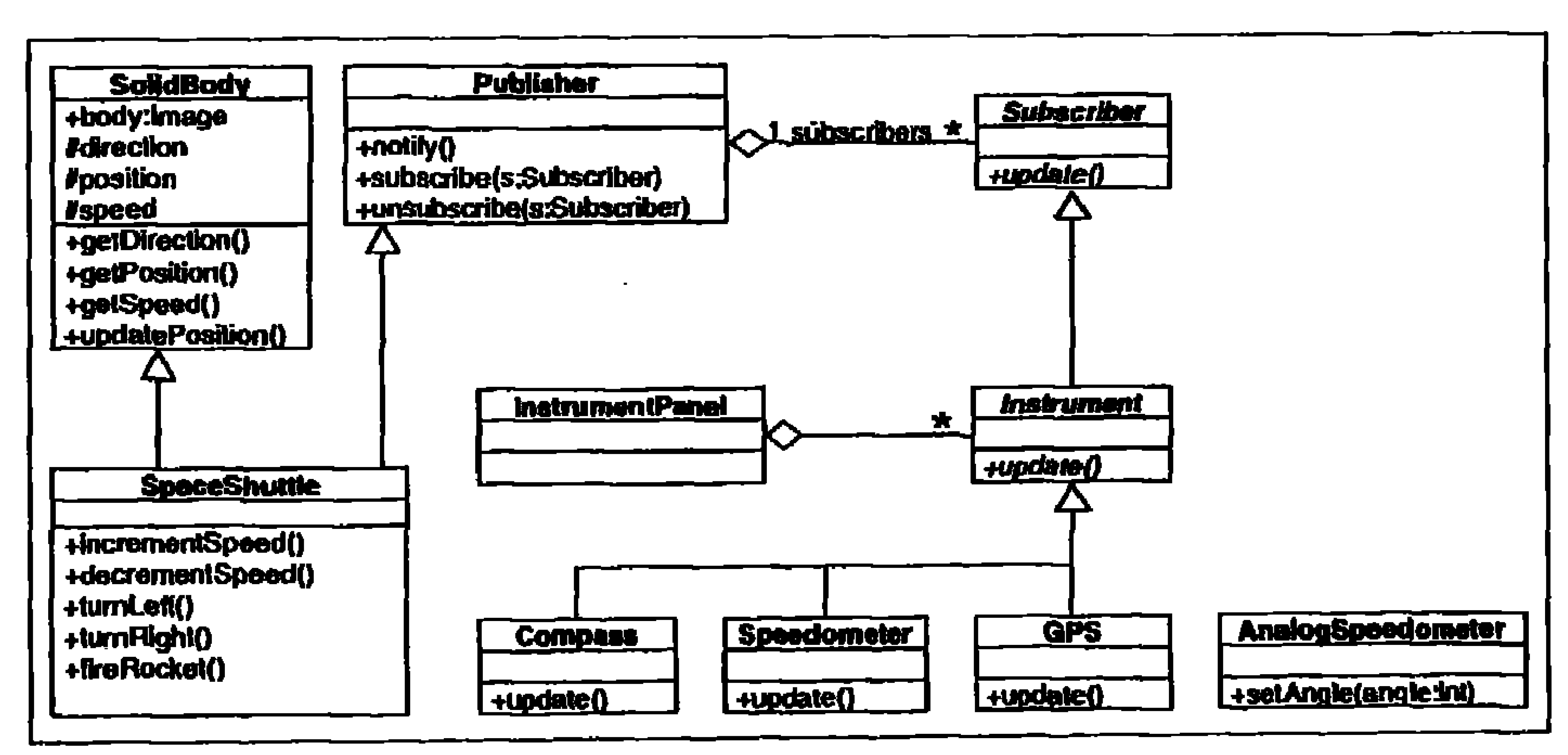
3 ـ 6: التمرين الثالث: نمذجة نمط الموائم

يركّز هذا التمرين على مسألة دمج مكوّنات البرمجية غير القابلة للتعديل التي لا تناسب تصميم النظام قيد التطوير. إما أن تكون هذه المكوّنات من مصادر خارجية أو نظم قديمة متوارثة توفر الوظائف المطلوبة. في التمرين الثالث، أضفنا النوع AnalogSpeedometer إلى مشروع لعبة الكويكبات وهذا النوع يعرض السرعة بواسطة إبرة مؤشر لكنه لا يزيد على النوع Instrument ولا يتبع Observer Pattern من التمرين الأول، كما لا يعرف النوع SpaceShuttle (انظر الشكل 3-5). يوفر النوع AnalogSpeedmeter العملية المشتركة setAngle (angle:int) التي تحدد زاوية أبرة عداد السرعة. وتكون قيمها من 0 إلى 180 درجة.

مهام التمرين

- ارسم مخطّط نوعٍ بلغة النمذجة الموحدة، ثم ادمج AnalogSpeedometer في آلية Observer Pattern (انظر الشكل 3-5)، بحيث تعرض سرعة المركبة الفضائية الحالية دائماً. يتم التعامل مع النوع AnalogSpeedometer على أنه نوع متوارث لا يمكن تعديله. أما نمط الموائم، فيجب أن يستخدم لعملية الدمج والتكامل (B. Bruegge and A. H. Dutoit, 2003[1]، ص 697).

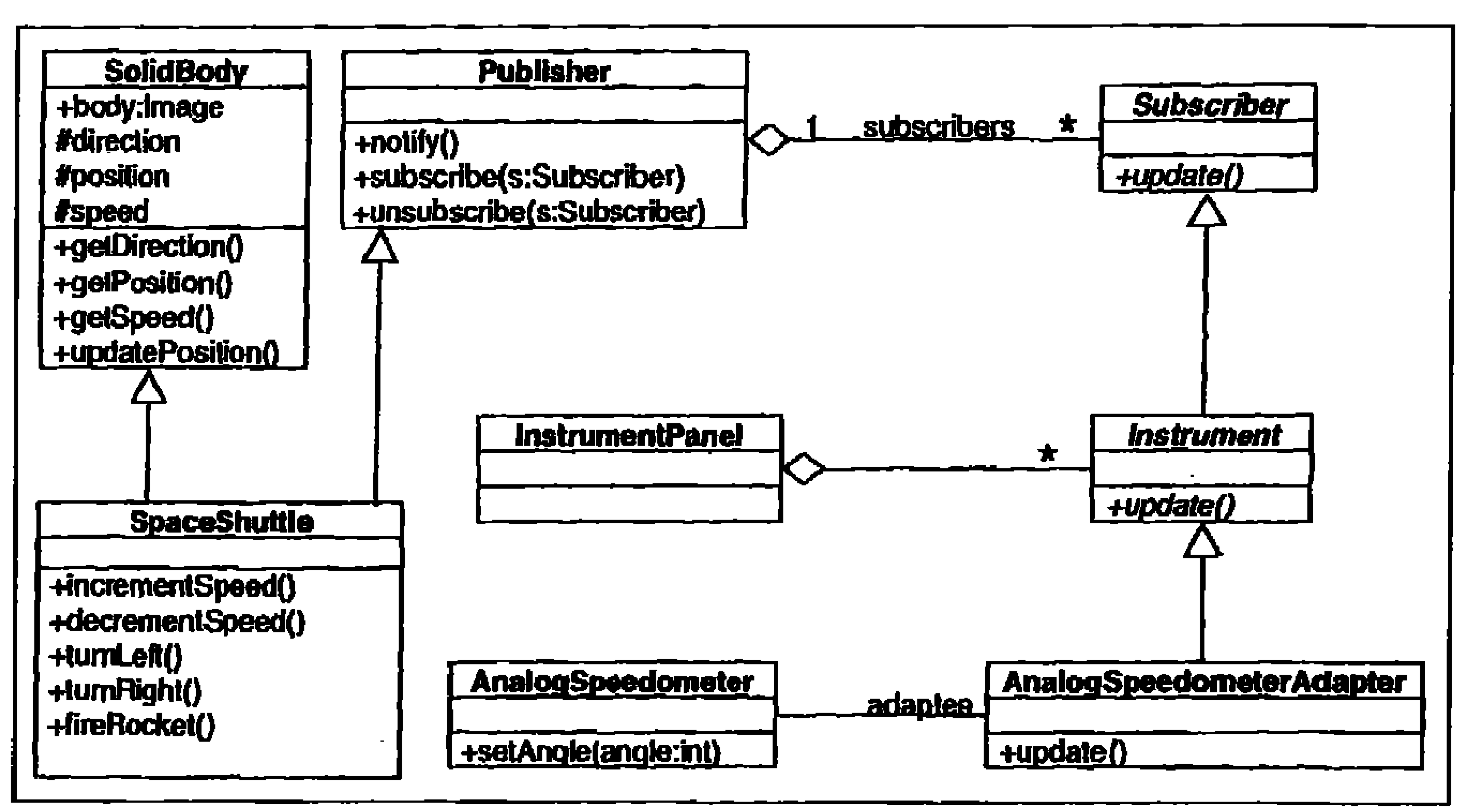
- ارسم مخطّط تسلسلٍ بلغةِ النمذجةِ الموحّدة، يوضح السلوك الديناميكي الذي يتبعه نمط الموائم عند تغيير AnalogSpeedometer.



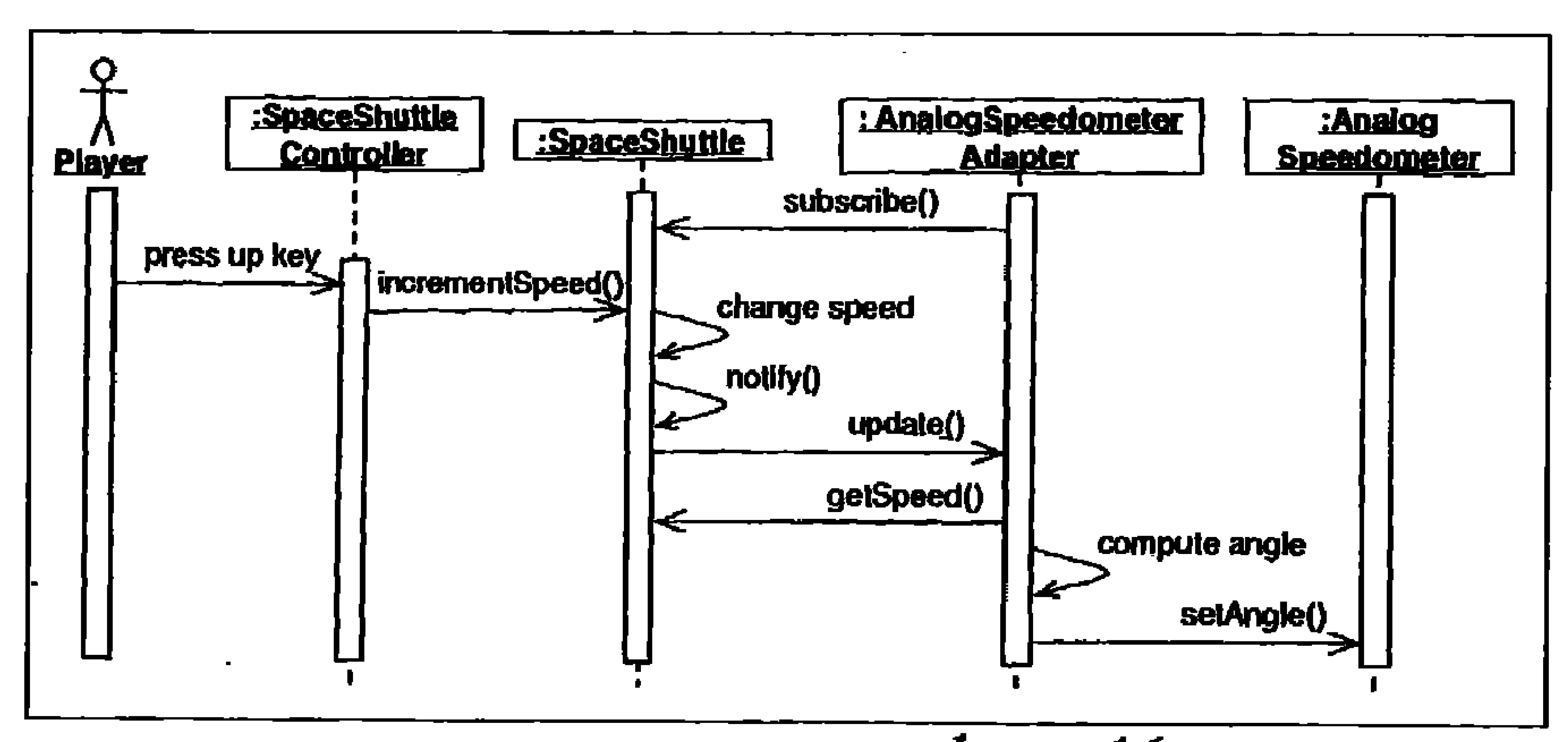
الشكل (3 ـ 5): مخطط يوضح النوع المتوارث AnalogSpeedometer الذي يجب أن يكون مدمجاً في تصميم اللعبة ولا يمكن تعديله

3 ـ 6 ـ 1 نموذج حل التمرين الثالث

يبيّن مخطط النوع بلغة النمذجة الموحّدة في الشكل 3-6 دمج AnalogSpeedometer في Asteroids باستخدام نمط الموائم. لقد حذفنا جميع الأدوات التي عرفناها مسبقاً لزيادة مستوى القراءة. يرتبط النوع الموائم مع AnalogSpeedometer ويزيد على النوع Instrument ليتناسب مع تصميم Observer Pattern



الشكل (3 ـ 6) : نموذج الحل لدمج النوع AnalogSpeedometer عن طريق تطبيق نمط الموائم



الشكل (3 ـ 7) : مخطّطٌ تسلسليٌ بلغة النمذجة الموحدة يوضح السلوك الديناميكي لـ AnalogSpeedometer المدمج بواسطة Adapter Pattern

يبيّن الشكل 3 ـ 7 مخطّطاً تسلسلياً بلغة النمذجة الموحدة، يوضح السلوك الديناميكي لعملية التكامل. النوع AnalogSpeedometerAdapter هو مشترك في المركبة الفضائية، ويتم استدعاء العملية update () عندما تتغير أي من خصائص المركبة. تسترجع العملية update () السرعة الحالية من المركبة، وتحسب الزاوية لعداد السرعة Analog Speedometer الذي يجب أن لا يتغير.

3 ـ 7 التمرين الرابع: برمجة Adapter Pattern

في التمرين الرابع، نقوم بتطبيق عملية تكامل Adapter Pattern الموضحة في القسم 3-6. هذا ويمكن تنزيل الشيفرة البرمجية للتمرين من خلال الموقع:
< http://$Asteroids/3_AdapterPatternExercise/downloads >

لقد أضفنا ملف نوع جافا المسمى AnalogSpeedometer.java إلى الحزمة legacysystem التي تتحقق من عداد السرعة التناظري. أما النوع الخاص بعداد السرعة التناظري Speedometer فيعمل على توسيع النوع JPanel وهو مُضاف أصلاً إلى المستوعب الجغرافي InstumentsPanel.

```
public class InstrumentPanel extends JToolBar {
:
public InstrumentPanel (SpaceShuttle theSpaceShuttle) {
:
analogspeedometer = new AnalogSpeedometer ();
add (analogspeedometer);
:
}
:
}
```

بعد تجميع وتنفيذ اللعبة، ستدرك أن عدّاد السرعة التناظري لا يظهر السرعة الحالية للمركبة الفضائية.

مهمة التمرين

- دمج عدّاد السرعة التناظري مع النظام باستخدام نمط التكيّف.
- إنشاء نوع جافا جديد باسم AnalogSpeedometerAdapter.java، حيث

يقوم هذا النوع بتطبيق نمط التكيّف، كما هو مبيّن في القسم 3-6. يجب أن ينقل السرعة من المركبة الفضائية إلى عداد السرعة التناظري. هذا ويجب أن لا يتغير الملف AnalogSpeedometer.java أبداً.

3 ـ 7 ـ 1 نموذج حل التمرين الرابع

يمكن تنزيل الشيفرة البرمجية الكاملة لحل التمرين الرابع من الموقع:

نقوم بإنشاء النوع AnalogSpeedometerAdapter في حزمة أدوات لعبة الكويكبات. تم توفير حالة النوع AnalogSpeedometer المكيف في المنظّم وتم إعداده على أنه متغير خاص للحالة. يعمل النوع AnalogSpeedometer المكيف على توسيع النوع Instrument. إن تطبيق ال method الخاصة بالتعديل update () يعمل على استرجاع السرعة الحالية للمركبة ويحسب زاوية عداد السرعة التناظري. لتقليل عمليات الحسابات غير المرغوب بها، نقوم بتخزين قيمة آخر سرعة للمركبة في متغير خاص بالحالة وحساب الزاوية فقط عندما تكون السرعة الحالية مختلفة.

```
public class AnalogSpeedometerAdapter extends Instrument {
private AnalogSpeedometer adaptee;
private int speed;
public AnalogSpeedometerAdapter (SpaceShuttle spaceshuttle,
AnalogSpeedometer analog_speedometer) {
super (spaceshuttle);
this.adaptee = analog_speedometer;
update ();
}
public void update () {
if (this.speed! = spaceshuttle.getSpeed ()) {
this.speed = spaceshuttle.getSpeed ();
double percent = 1.0d/(double) spaceshuttle.getMaximumSpeed ()
* (double) this.speed;
int angle = (int) ((double) adaptee.getMaxAngle () * percent);
this.adaptee.setAngle (angle);
}
}
}
```

لتمكين آلية التكيّف، نقوم بتعديل النوع InstrumentPanel عن طريق إنشاء حالة AnalogSpeedometerAdapter جديدة، ومن ثم الاشتراك بها في المركبة الفضائية.

```
public class InstrumentPanel extends JToolBar {
:
private AnalogSpeedometer analogspeedometer;
private AnalogSpeedometerAdapter analogspeedometeradapter;
:
public InstrumentPanel (SpaceShuttle theSpaceShuttle) {
:
analogspeedometer = new AnalogSpeedometer ();
analogspeedometeradapter = new AnalogSpeedometerAdapter (spaceshut-
     tle,
analogspeedometer);
theSpaceShuttle.subscribe (analogspeedometeradapter);
add (analogspeedometer);
:
}
:
}
```

3 ـ 8 التمرين الخامس: نمذجة نمط الاستراتيجية

في نظام لعبة الكويكبات يكون النوع Referee مسؤولاً عن تحديد التصادمات بين الأجسام الصلبة المختلفة. بعد فترة زمنية، يحرّك النوع Referee كل جسم من الأجسام ويحدد أيّاً من الأجسام الصلبة يتقاطع مع غيره. يستخدم هذا النوع detectCollision () ـ المبين في شيفرة البرمجة المصدرية الآتية ـ لتحديد التقاطع بين جسمين. تأخذ هذه ال method كائنين من نوع SolidBody كعوامل. إذا حدث التقاطع، يتم استرجاع الجسم المدمَّر SolidBody؛ أما إن لم يحصل التقاطع، يتم استرجاع القيمة null. تستخدم الأنواع الفرعية للعوامل ـ وهي Astroids، SpaceShuttle، Rocket ـ لتحديد ما إذا تحطم الجسم. إن استراتيجية التصادم المطبقة سهلة للغاية: إذ تتحطم المركبة الفضائية عندما تتقاطع مع أي جسم آخر، أما إذا تقاطعت الكويكبات

مع بعضها بعضاً فإنها لا تتحطم، أما الصاروخ فيعمل على تدمير الأجسام الأخرى، لكنه لا يُحطم ذاته.

```
public SolidBody detectCollision (SolidBody solidbody1, SolidBody
solidbody2) {
Point p1 = GameBoard.getInstance ().convertPosition(
solidbody1.getPosition ());
Dimension d1 = solidbody1.getSize ();
Rectangle r1 = new Rectangle (p1, d1);
Point p2 = GameBoard.getInstance ().convertPosition(
solidbody2.getPosition ());
Dimension d2 = solidbody2.getSize ();
Rectangle r2 = new Rectangle (p2, d2);
if (r1.intersects (r2)) {
if (solidbody1 instanceof SpaceShuttle) {
return solidbody1;
} else if (solidbody2 instanceof SpaceShuttle) {
return solidbody2;
} else if (solidbody1 instanceof Rocket) {
return solidbody2;
} else if (solidbody2 instanceof Rocket) {
return solidbody1;
} else {
return null;
}
}
return null;
}
```

لهذا التطبيق عدة مساوئ: الشيفرة البرمجية للعملية () detectCollision صعبة القراءة، لأنها تحتوي على جمل شرطية في العديد من المستويات المتداخلة. إضافة إلى ذلك، يتم إدراك استراتجية التصادم خارج النوع SolidBody وأنواعه الفرعية. كنتيجة ذلك، من الصعب إضافة متطلبات جديدة، على سبيل المثال:

يجب أن يكون لاعب لعبة الكويكبات قادراً على تغيير استراتيجية تصادم المركبة الفضائية في أثناء فترة التنفيذ.

حتى تكون قادراً على التعامل مع أنواع المتطلبات هذه، نقوم بعمل مراجعة للشيفرة البرمجية المصدرية وتحولات النموذج في مخطط النوع:

1. مراجعة الشيفرة البرمجية المصدرية. نستخلص وظيفة التحقق ممّا إذا كانت الأجسام الصلبة تتحطم، وننقلها إلى عملية جديدة method تدعى:

+ collide (body:SolidBody):boolean of the class SolidBody

وبناء على ذلك، سيكون على Referee اكتشاف إذا تقاطع جسمان صلبان، ويستعلم ما إذا تحطم الجسمان وذلك عن طريق استدعاء العملية collide ().

يبيّن المقطع الآتي من الشيفرة البرمجية التغييرات التي تحدث في النوع Referee.

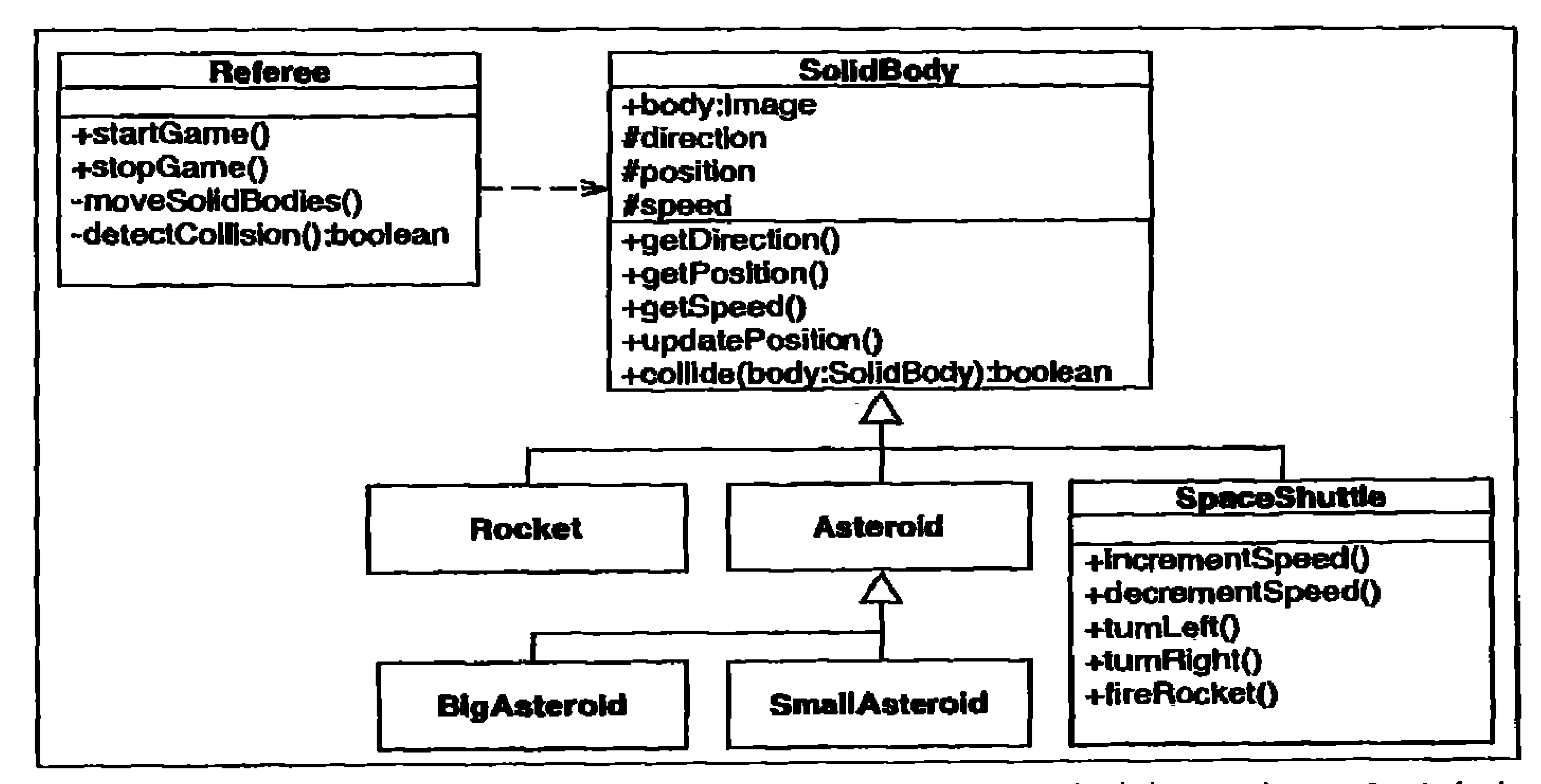
```
private void moveSolidBodies () {
GameBoard gameBoard = GameBoard.getInstance ();
SolidBody[] solidbodies = gameBoard.getSolidBodies ();
int max_x = gameBoard.getSize ().width;
int max_y = gameBoard.getSize ().height;
for (int i = 0; i < solidbodies.length; i++) {
solidbodies[i].updatePosition (max_x, max_y);
}
gameBoard.repaint ();
HashSet<SolidBody> crashedBodyCache = new HashSet<SolidBody> ();
for (int z = 0; z < solidbodies.length; z++) {
SolidBody solidbody1 = solidbodies[z];
if (crashedBodyCache.contains (solidbody1)) {
continue;
}
for (int i = 0; i < solidbodies.length; i++) {
SolidBody solidbody2 = solidbodies[i];
if (solidbody1 == solidbody2) {
continue;
}
if (crashedBodyCache.contains (solidbody2)) {
```

```
continue;
}
boolean collision = detectCollision (solidbody1, solidbody2);
if (collision) {
boolean isCrashed = solidbody1.collide (solidbody2);
60 TEACHING DESIGN PATTERNS
if (isCrashed) {
crashedBodyCache.add (solidbody1);
GameBoard.getInstance ().removeSolidBody (solidbody1);
}
isCrashed = solidbody2.collide (solidbody1);
if (isCrashed) {
crashedBodyCache.add (solidbody1);
GameBoard.getInstance ().removeSolidBody (solidbody2);
}
}
}
}
if (!gameBoard.hasSolidBody (gameBoard.getSpaceShuttle ())) {
stopGame ();
JOptionPane.showMessageDialog (null, "You lost the game in "
+ gameduration_in_seconds + " seconds!", "Information",
JOptionPane.INFORMATION_MESSAGE);
int index = (int) (Math.random () * (double) gameoverClips.size ());
AudioClip clip = (AudioClip) gameoverClips.get (index);
clip.play ();
initGame ();
} else if (GameBoard.getInstance ().getSolidBodies ().length = = 1) {
stopGame ();
JOptionPane.showMessageDialog (null,
"Congratulation, you won the game in "
+ gameduration_in_seconds + " seconds!",
"Information", JOptionPane.INFORMATION_MESSAGE);
initGame ();
}
}
public boolean detectCollision (SolidBody solidbody1, SolidBody
```

```
solidbody2) {
Point p1 = GameBoard.getInstance ().convertPosition(
solidbody1.getPosition ());
Dimension d1 = solidbody1.getSize ();
Rectangle r1 = new Rectangle (p1, d1);
Point p2 = GameBoard.getInstance ().convertPosition(
solidbody2.getPosition ());
Dimension d2 = solidbody2.getSize ();
Rectangle r2 = new Rectangle (p2, d2);
if (r1.intersects (r2)) {
return true;
} else {
return false;
}
}
```

2. التحول في النموذج. يبيّن الشكل 3 ـ 8 مخطط النوع بلغة النمذجة الموحدة حسب التغيرات الموضحة أعلاه. يستدعي النوع Referee العملية (...collide) ضمن الأحداث الخاصة بالأجسام الصلبة SolidBody عندما تكون نتيجة عملية التحقق () detectCollision إيجابية.

لاحظ أننا نعرض الأنواع ذات العلاقة فقط.



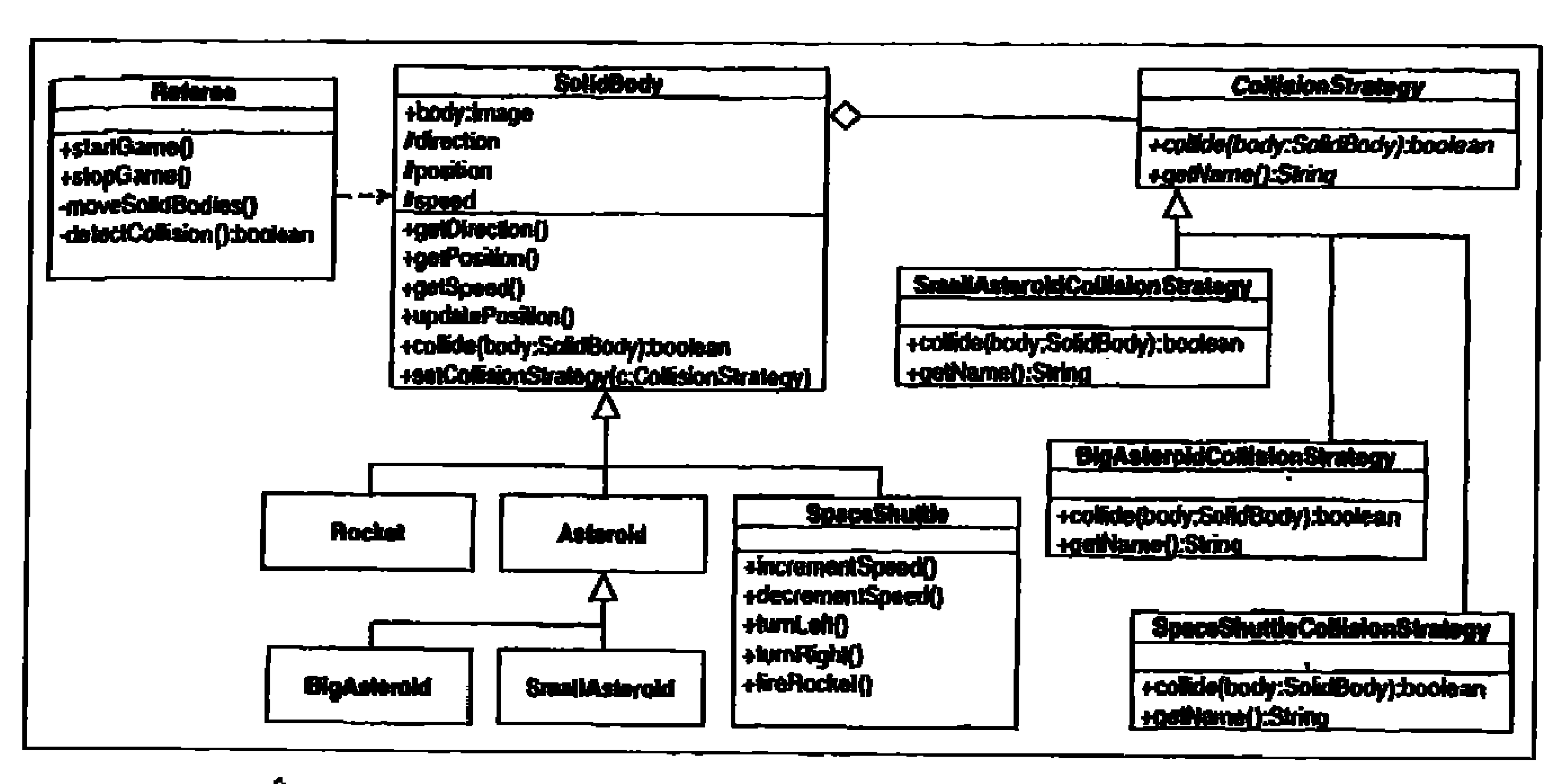
الشكل (3 ـ 8): مخطط الصنف UML يبين التغيرات المراجعة، ويقتصر الشكل على الأصناف ذات العلاقة

مهمة التمرين

لتحقيق المتطلب «تغيير استراتيجية التصادم للمركبة الفضائية في فترة التنفيذ»، فإن مهمة التمرين تطبيق نمط الاستراتيجية في مخطط النوع بلغة النمذجة الموحدة.

- يجب أن يتم إدراك استراتيجية التصادم في نوع منفصل يدعى CollisionStrategy . هذا النوع نظري فحسب ويجب أن يوفر العمليات النظرية التي تحدد وظيفة عملية (...) collide بحيث يتمكن النوع SolidBody من تفويض النوع CollisitionStrategy بإداء جميع طلبات التنفيذ.
- يوفر استراتيجيات تصادم متمكنة للمركبة الفضائية والأنواع الخاصة بالكويكبات.
- توفير عمليات لتغيير استراتيجيات التصادم في فترة التنفيذ.

3 ـ 8 ـ 1 نموذج حل التمرين الخامس



الشكل (3 ـ 9) : مخطط الصنف UML يبين فصل (Decoupling) الأجسام الصلبة واستراتيجيات ارتطامها باستخدام نمط الاستراتيجية

يبيّن مخطط النوع بلغة النمذجة الموحدة في الشكل 3 ـ 9 استخدام نمط الاستراتيجية للتحقق من حدوث تغيير في استراتيجيات التصادم في أثناء فترة التنفيذ. يستدعي النوع Referee العملية (...) collide من النوع SolidBody وهذا النوع يفوض CollisionStrategy المرتبط بتنفيذ الطلب call. يُجيب النوع الفرعي

الخاص بتنفيذ استراتيجية التصادم الحالية، ومن ثم يعيد النوع SolidBody نتيجة التنفيذ إلى Referee. النوع الفرعي غير معروف للنوع SolidBody وقد يتغير في أثناء فترة التنفيذ بواسطة العملية (...) setCollisionStrategy. لقد أضفنا العملية getName ():String التي تعطي عن تنفيذها اسم استراتيجية التصادم الذي سيستخدم لعرض استراتيجية التصادم الحالية. الآن، تم توسيع لعبة الكويكبات لتشمل استرتيجيات تصادم جديدة من دون تعديل النوعين Referee وSolidBody.

3 ـ 9 التمرين السادس: برمجة نمط الاستراتيجية

يرتكز التمرين السادس على شيفرة تحويل العوامل للعبة الكويكبات، التي يمكن تنزيلها من الموقع: <http://$Asteroids/4_StrategyPatternExercise/ downloads>

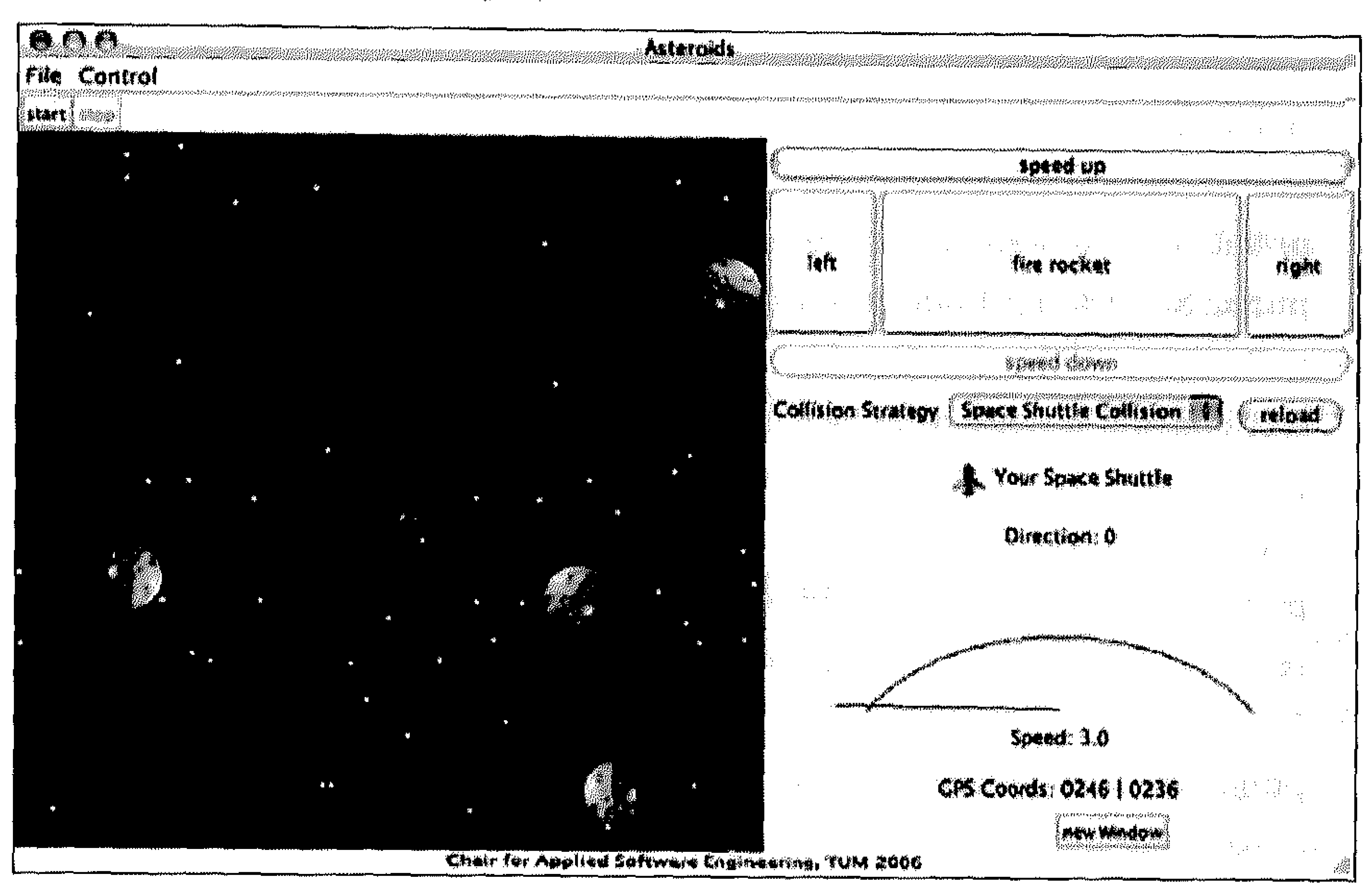
لقد نفّذنا التصميم المبيّن في القسم 3-8. توجد أنواع نمط الاستراتيجية ضمن الحزمة collisionstrategies. تبيّن الشيفرة الآتية النوع النظري collisionstrategy. يتطلب تنفيذ الأنواع الفرعية مرجعاً للأجسام الصلبة التي تستدعي العملية (...) collide لأجلها. وبناء على ذلك، يوفر النوع CollisionStrategy متغير خاص ذو عمليات «جلب» (getter) و«تحديد» (Setter) مناسبة.

```
public abstract class CollisionStrategy {
private SolidBody solidBody;
public void setSolidBody (SolidBody solidBody) {
this.solidBody = solidBody;
}
public SolidBody getSolidBody () {
return solidBody;
}
public abstract boolean collide (SolidBody opponent);
public abstract String getName ();
public String toString () {
return getName ();
}
}
```

يبيّن المقطع الآتي من الشيفرة البرمجية النوع SolidBody. هذه الشيفرة تفوض استراتيجية التصادم الحالية بتنفيذ التصادم وتوفر العمليات اللازمة لاسترجاع وتغيير الاستراتيجية. يتم استدعاء العملية (...) setSolidBody في نوع استراتيجية التصادم من العملية (...) setCollisionStrategy في النوع SolidBody.

```
public abstract class SolidBody {
:
private CollisionStrategy collisionStrategy;
public SolidBody (Point position, int direction) {
:
setCollisionStrategy (new DefaultCollision ());
}
:
public boolean collide (SolidBody solidbody) {
return collisionStrategy.collide (solidbody);
}
public void setCollisionStrategy (CollisionStrategy
collisionStrategy) {
if (collisionStrategy = = null) {
throw new IllegalArgumentException(
"The given collision strategy is null.");
}
if (this.collisionStrategy! = null) {
this.collisionStrategy.setSolidBody (null);
}
this.collisionStrategy = collisionStrategy;
this.collisionStrategy.setSolidBody (this);
}
public CollisionStrategy getCollisionStrategy () {
return collisionStrategy;
}
:
}
```

بعد تجميع لعبة الكويكبات وتنفيذه، ستظهر قائمة أوامر ضمن لوحة التعلميات (انظر اللقطة في الشكل 3-10). تعرض القائمة استراتيجية تصادم المركبة الفضائية الحالية وتتضمن جميع الاستراتيجية الموجودة ضمن حزمة collisionstrategies التي يتم تحميلها ديناميكياً عن طريق استخدام Java Reflection. هذا ويمكن تغيير استراتيجية التصادم في فترة التنفيذ.



الشكل (3 ـ 10): لقطة لشاشة لعبة الكويكبات تتضمن اختيار استراتيجية التصادم

مهمة التمرين

- تنفيذ استراتيجية تصادم جديدة تسمح للمركبة الفضائية التصادم مع ثلاثة كويكبات قبل أن تتحطم. يجب أن تكون الاستراتيجية الجديدة متوفرة ضمن الحزمة collisionstrategies. وبخلاف ذلك، لن تعثر آلية تحميل النوع الديناميكية من العثور عليها.

3 ـ 9 ـ 1 نموذج حل التمرين السادس

يمكن تنزيل الشيفرة البرمجية المصدرية الكاملة لنموذج حل التمرين السادس من الموقع:

لقد نفّذنا النوع SpaceShuttle3Credits الذي يوسع النوع Space Shuttle

Collision Strategy ويعيد استخدام تنفيذ العملية (...) collide التابعة له. للتحقق من المتطلبات المطلوبة، أضفنا عدّاداً يدعى credits وهو يتفعل بالقيمة 3. تقل قيمة العداد عندما تكون نتيجة تنفيذ العملية (...) collide إيجابية. هذا وتكون نتيجة تنفيذ النوع SpaceShuttle3Credits خاطئة إلى أن تصل قيمة العداد إلى 0. يتطلب تقاطع جسمين صلبين بعض الوقت قبل أن ينفصلا ثانيةً. لتجنب انخفاض قيمة العداد إلى 0 ضمن حالة تقاطع واحدة، نقوم بتقليل قيمة العداد خلال فواصل زمنية، مدة كلٍّ منها ثلاث ثوانٍ.

```
public class SpaceShuttle3Credits extends
SpaceShuttleCollisionStrategy {
private int credits;
private long lastCrashInMillis;
public SpaceShuttle3Credits () {
super ();
credits = 3;
lastCrashInMillis = System.currentTimeMillis ();
}
public boolean collide (SolidBody opponent) {
boolean isCrashed = super.collide (opponent);
if (isCrashed && credits > 0) {
if ((System.currentTimeMillis
-lastCrashInMillis)/1000 > 3) {
lastCrashInMillis = System.currentTimeMillis ();
credits–;
}
return false;
} else {
return isCrashed;
}
}
public String getName () {
return "3 credits";
}
}
```

3 ـ 10 الخبرات والاستنتاجات

لقد طبقنا المنهج الذي زودنا به على أنماط التصميم مرات عديدة في بيئات تعليمية مختلفة. استخدمنا التمارين في محاضرات هندسة البرمجيات في جامعة ميونيخ التكنولوجية في فصول الشتاء للأعوام الدراسية 2005/2004 و2006/2005. حضر حوالى 100 طالب، من الفصل الثالث، دروساً عملية أسبوعية، نفّذنا خلالها تمارين النمذجة بشكل تفاعلي مع الطلاب. كنا نناقش المشكلة ثم نطلب من الطلاب كتابة الحل على الورق، وكنا نقدم المساعدة الفردية لهم. ومن ثم كان أحد الطلاب يعرض نموذج الحل خاصته قبل أن نقدم لهم الحل ونناقشه. كان على الطلاب إنجاز تمارين البرمجة كلّ على حِدَة خلال الأسبوع.

لقد نفذنا تمارين لعبة الكويكبات ضمن منهج دراسي عن التطوير الذي تحكمه الأنماط خلال الدورة الصيفية الدولية التي عقدت عن هندسة البرمجيات[5]. لقد حضر طلاب الشهادة الجامعية العليا، الدكتوراه (Ph.D)، المحاضرة ذات الأربع ساعات التي تضمنت معرفة عميقة بأنماط التصميم والتطوير كائني التوجه والبرمجة.

استخدمت باتريشيا لاغو (Patricia Lago) التمارين لتدريس أنماط التصميم ضمن منهاج هندسة البرمجيات في جامعة Vrije أمستردام في ربيع عام 2006[3]. يركّز منهاج هندسة البرمجيات على المبادئ النظرية لهندسة البرمجيات ويغطي أنماط التصميم في أسبوع واحد فقط. شارك حوالى 120 طالب من السنة الثانية في درجة البكالوريوس علوم الكمبيوتر في محاضرة، وقسموا إلى مجموعات تضمنت كل منها 30 طالباً. نفذت كل مجموعة من المجموعات مادة الدورة في مختبر الحاسوب. بدأ المنهاج بعرض مدته 15 دقيقة عن أنماط التصميم بشكل عام. ثم تم تقديم كل تمرين بما في ذلك نمط التصميم المرتبط به لمدة 15 دقيقة، ومن ثم قاموا بحل كل تمرين على حِدَة في مدة 30 دقيقة. تلقينا انطباعات إيجابية عن تنفيذ تلك المادة التعليمية.

لقد كانوا قادرين على تعليم أنماط التصميم بما في ذلك الخبرات العملية ضمن فترة محدودة وحققوا نجاحاً في تدريس أنماط التصميم للطلاب.

لقد لاحظنا أن تدريس أنماط التصميم الذي يعتمد على نظام حقيقي بدلاً من الاعتماد على أمثلة صغيرة يحسّن من فهم الطلاب للتطوير المرتكز على أنماط التصميم.

إن استخدام النظم الكبيرة يزيد من الصعوبات في البداية، لكنه يزيد من تحفيز الطلاب وإبداعهم أيضاً، عند إدراك القدرة على فهم النظام وتوسيعه بمعرفة مفاهيمه. لقد استلمنا العديد من الحلول التي تخطت المهام المطلوبة وزاد طلب الطلاب لفهم خصائص إضافية للعبة الكويكبات.

المراجع

1. B. Bruegge and A. H. Dutoit. *Object-Oriented Software Engineering Using UML, Patterns, and Java.* 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 2003.

2. E. Gamma, R. Helm, and R. Johnson. *Design Patterns. Elements of Reusable Object-Oriented Software.* Addison-Wesley Professional Computing Series, Reading, MA: Addison-Wesley, 1995.

3. P. Lago, R. Farenhorst, and R. de Boer. *Software Engineering, Vrije Universiteit Amsterdam*, Spring 2006. <http://bb.vu.nl/webapps/portal/frameset.jsp?tab=courses&url=%2Fbin%2Fcommon%2Fcourse.pl%3Fcourse_id%3D_17030_>.

4. A. Schmolitzky. «A laboratory for teaching object-oriented language and design concepts with teachlets.» paper presented at: *Proceedings of the Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA '05) Companion to the 20th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, 2005, pp. 332-337.

5. Second International Summer School on Software Engineering. September 2005, Salerno. <http://www.sesa.dmi.unisa.it/seschool/previousEditions/2005/>.

الجزء الثاني

الأساليب الحديثة

4

تأثير هندسة البرمجيات أدواتية التوجه في الحوسبة خدمية التوجه

لورا بوتشي (Laura Bocchi)
باولو تشانكاريني (Paolo Ciancarini)
روكو موريتي (Rocco Moretti)
فالنتينا بريشوتي (Valentina Presutti)

4 ـ 1 المقدمة

نناقش في هذا الفصل تأثير هندسة البرمجيات أدواتية التوجه (AOSE) في الحوسبة خدمية التوجه. نأخذ في الاعتبار بعض الأفكار الأساسية للهيكلة خدمية التوجه (SOA)[7] في سياق التقنيات الأدواتية وذلك لتنسيق وتركيب الخدمات الشبكية وخدمات الويب.

في العقد الماضي، تطورت شبكة الإنترنت تماشياً مع تطور خدمات الإنترنت المهيمنة لتصبح الأكثر شيوعاً والأوسع انتشاراً لنظم المعلومات في العالم أجمع. أما صلب هذه الشبكة فهو النصوص المتشعبة hypertext التي تعرض بها الوثائق من قبل الأجهزة الخادمة servers وبواسطتها تسترجع الأجهزة التابعة clients الوثائق بالاستعانة ببروتوكول نقل النصوص التشعبية HTTP وتعرض من خلال واجهات الاستخدام الرسومية سهلة الاستخدام. نظراً إلى انتشارها الواسع، أصبح بالإمكان اعتماد شبكة الإنترنت كمنصة إطلاق للعديد من التطبيقات الديناميكية الموزعة سواء كانت تطبيقات صناعية أو أكاديمية[11].

في الوقت ذاته، من أهم الاتجاهات التي نلحظها اليوم في مجالات علوم

الحاسوب المختلفة إنشاء نظم معقدة مكوّنة من أجزاء أبسط وغير متجانسة وموزعة. في حالة الأعمال الإلكترونية، لاحظنا اتجاهاً عاماً نحو الاستعانة بالمصادر الخارجية[24]. والاستعانة بالمصادر الخارجية تعني التعاقد مع عاملين من خارج الشركة لتنفيذ مهام محددة بدلاً من استخدام الموظفين. إن تطبيق هذا المصطلح على البنية التحتية لتكنولوجيا المعلومات لشركة ما يعني استغلال الموارد الخارجية من أجهزة وبرمجيات ودمجها بالنظام الداخلي للشركة. وعليه فقد تتكون البنية التحتية لتكنولوجيا المعلومات للشركة من الموارد الخارجية والشبكات والخدمات، إضافة إلى الأجزاء الداخلية أو المتوارثة.

بالنسبة إلى العلوم الإلكترونية، فإن ثمة حاجة كبيرة إلى الطاقة الحسابية ووسائل التخزين الكبيرة لإدارة كميات البيانات الكبيرة وذلك لدعم المهام والتجارب العلمية. منذ أواسط عقد التسعينيات من القرن الماضي، عُنيت نماذج التصميم الشبكية الأولى بمعالجة هذه القضية. إن الحاجة إلى وسيط يعمل على تكامل الخدمات الديناميكية التي يتم تشغيلها ضمن منصات موزعة وغير متجانسة قادت إلى نشوء حلول مختلفة لكنها متداخلة في بعض أجزائها.

من الصعوبة بمكان تحديد إطار عملي عام يتيح إدارة الخدمات بطريقة قياسية معيارية مستقلة. قد يكون الوسيط الذي يدعم التطبيقات الموزعة نقطة بداية فاعلة لتنفيذ إطار عملي كهذا. هذا ولم تتحقق التوافقية الكاملة بين منصات الوسائط المختلفة[26]. هناك حاجة إلى إطار عملي خفيف يمكن أن يحقق التوافقية بين تقنيات الوسائط المختلفة المعروفة حالياً، وذلك لدعم الخدمات ضمن بنية مرجعية موحدة.

تحاول كلٌ من منهجية الهيكلية القائمة على النماذج (MDA)[26] والهيكلية خدمية التوجه أن تعالج هذه الاحتياجات. تبدأ الهيكلية القائمة على النماذج من الوحدات ومواصفاتها، بينما تبدأ الهيكلية القائمة على الخدمات من منهجية الحوسبة الموزعة التي تأخذ بالاعتبار الموارد البرمجية والخدمات المتاحة على الشبكة[7].

نحن نؤكد الأسباب التي تجعل من الأدوات (agents) حلولاً ملائمة لتصميم وتطوير النظم خدمية التوجه عبر شبكة الإنترنت والبنية التحتية للشبكية. في هذا السياق، نركز على دور هندسة البرمجيات أدواتية التوجه (AOSE) في هيكلية خدمات الويب (WSA)[5] والشبكية.

يتبنّى العديد من نطاقات التطبيقات المختلفة بروتوكولات الإنترنت كأساس

للبنى التحتية للوسائط خدمية التوجه. مثلاً، تركز الأعمال الإلكترونية والعلوم الإلكترونية على منصات الوسائط التي تدمج الفكرة العامة للهيكلة خدمية التوجه (SOA) باستخدام البنى التحتية للإنترنت والبروتوكولات الخاصة بها، أي إنها تستخدم هيكلية خدمات الويب المستخدمة للأعمال الإلكترونية وتستخدم هيكلية خدمات الشبكية المفتوحة (OGSA)[21] للعلوم الإلكترونية.

تم تنظيم هذا الفصل على النحو الآتي: يقدّم القسم 4-2 للنظم الأدواتية وهندسة البرمجيات أدواتية التوجه، بينما يعرض القسم 4-3 لمحة عامة عن الحوسبة خدمية التوجه AOSE. يعرض القسم 4-4 قضايا تتعلق بالخدمات القائمة على النماذج MDA للأدوات الشبكية. يركّز القسم 4-5 على التنسيق في هيكلية خدمات الويب. يناقش القسم 4-6 مسألة تحديد توصيف لتوفير أدوات في هيكلية خدمات الويب بطريقة قابلة للقراءة في مجال الاهتمام. أخيراً يعرض القسم 4-7 استنتاجاتنا مع الإشارة إلى الاتجاهات البحثية الممكنة في المستقبل.

4 ـ 2 النظم الأدواتية وهندسة البرمجيات أدواتية التوجه

الأداة (agent) هي «نظام حاسوب مغلف يتواجد ضمن بيئة معيّنة بحيث يكون قادراً على التصرف الذاتي بمرونة في تلك البيئة وذلك بهدف تحقيق الأهداف التي صمم من أجلها»[37]. من هذا التعريف، يمكننا القول إن النظام الأداة هو طريقة للتفكير (استعارة برمجية) في النظم التي تتكون من كيانات فاعلة (أي من أدوات) وفي سلوكها المشترك. هذا وقد تكون النظم الأدواتية متباينة جداً، وقد تتضمن أجهزة ومعدات وبرمجيات ووثائق سارية المفعول وخدمات متناسقة وشبكات كاملة وأفراد.

يكون استخدام «الأداة» فاعلاً أكثر ما يكون في بناء برمجيات خاصة بالنظم الشبكية المعقدة حيث لا يمكن أن تتوفر سيطرة شاملة. يعتمد قرار استخدام أداة واحدة أو عدة أدوات على مدى استخدام الوحدات وعلى الخاصية التي يرغب في إحرازها عند تصميم النظم المعقدة. في هذا السياق، فإن الأداة هي أداة برمجة تعتمد على نموذج حوسبة محدد يرتبط بنماذج أخرى كالوظائف الفرعية والوظائف المشاركة والإجراءات والعمليات والأنواع/الكوائن. عموماً، يستخدم المصطلح البرمجي «أداة agent» بطرق عديدة: كعمليات/برامج خفية دائمة أو شيفرة متغيرة أو روبوتات ذاتية التحكم أو أدوات ذكية (أي لا يوجد اتفاق حول ما يجعل منها أدوات ذكية).

في هذا الفصل، ما نعنيه بالأدوات البرمجية هو وحدات البناء المنطقية للجيل القادم من الوسائط. إذ ستبني هذه الوسائط فوق الوسائط المستخدمة حالياً (مثال، CORBA, EJB, Jini) وستوفر تكاملاً في أثناء فترة التنفيذ من خلال الاكتشاف الديناميكي والتفاوض على الموارد. في هذا السياق، نعتبر أن هندسة البرمجيات أدواتية التوجه هي نظام يتعامل مع تصميم وتطوير التطبيقات الموزعة ذات الأدوات المتعددة[12].

تركز هندسة البرمجيات أدواتية التوجه على العلاقات بين المكوّنات وعلى هيكلية البرمجية ومنهجية التضميم المستخدمة لتطوير التطبيقات ذات الأدوات المتعددة. يتم تحديد أطر عمل ملائمة لبناء نظم متماسكة ذات بنية جيدة من مكوّنات مفردة ولفهم التطبيقات المعقدة ذات الأدوات المتعددة وإدارتها وصيانتها.

توفّر هندسة البرمجيات أدواتية التوجه أساساً مفاهيمياً ذا جذور ممتدة في نطاقات المشكلة ذلك أن الأدوات في طبيعتها نظرية (تقليدية) في العديد من السياقات. ومن الحوافز الأخرى لهندسة البرمجيات أدواتية التوجه القدرة على زيادة إمكانيات التحويل إلى المواصفات المحلية للمستخدم وإخفاء تفاصيل مكوّنات التطبيقات التي تعمل من خلال شبكة الإنترنت والدعم القوي لإعادة استخدام التصاميم والبرامج القديمة. تتيح هندسة البرمجات أدواتية التوجه أيضاً استخدام مكوّنات النظم الفرعية كاملةً (بما في ذلك أنماط التصميم والمكوّنات الجاهزة التجارية)، أي الهيكليات المتعددة الأدوات والتفاعلات المرنة ما بينها (أطر عمل التطبيقات) إضافة إلى هيكليات التنسيق وبروتوكولات المزاد (Auction Protocols).

بدءاً من المنهجية القائمة على هندسة البرمجيات أدواتية التوجه، يمكن تعريف دورة حياة البرمجية أدواتية التوجه كما يأتي:

1. مواصفات متطلبات النظم الأدواتية.
2. تحليل المتطلبات.
3. التصميم.
4. كيفية تنفيذ هذه النظم.
5. كيفية التحقق من أن النظم المنفذة تحقق المواصفات المطلوبة.

غالباً ما يتم توفير مواصفات المتطلبات من قبل المستخدم من ناحية السيناريوهات المرغوب بها أو غير المرغوب بها. عادة ما تكون هذه المتطلبات غامضة ومتناقضة وغير كافية لتصميم النظام. إن الهدف من تحليل المتطلبات هو تحديد سلوك النظام الإجمالي المرغوب به. ويتم التركيز في هذه المرحلة على أهداف النظام الأساسية والأدوار التي يجب أن يقوم بها لحل المشكلة والموارد المتوفرة للحل والتفاعل مع المستخدمين والمتطلبات. تحديداً، يتم تحديد الأمور التي قد تختلف عن المتطلبات لتحقيق السلوك المتوقع وما لا يمكن قبول اختلافه. أما مرحلة التصميم فالهدف منها تحويل الأهداف والأدوار إلى أدوات واقعية، أي تحديد الأدوات التي ستنفذ أدواراً معيّنة. في هذه المرحلة، يتم تحديد أنواع الأدوات وعددها، إضافة إلى تحويل التفاعلات المتقدمة إلى بروتوكولات تفاعل محددة؛ كما يتم في هذه المرحلة تحديد نوع الأداة لكل وظيفة عادية مطلوبة من النظام حتى يتم تنفيذ السلوكيات المتقدمة للنظام. في مرحلة التنفيذ، يتم تحويل البروتوكولات والسلوكيات العادية إلى شيفرة برمجية. إضافة إلى ذلك، يتم أخذ قرار بشأن آلية ولغة ومضمون التواصل ولغة التنفيذ. أما بالنسبة إلى مرحلة التحقق، فيتم تنفيذ عملية لعرض دليل على تحقيق مواصفات المتطلبات.

يبدو أن الأساليب أدواتية التوجه المستخدمة في تطوير البرمجيات التي تحتل حيّزاً كبيراً في مجتمعات البحث[10، 36، 40] أصبحت نقطة بداية ذات معنى نحو تعريف مناهج تطوير البرمجيات خدمية التوجه. حتى نعرض هذا التطور الممكن، عرضنا باختصار منهجين من هذه المناهج، هما Gaia وTropos.

منهجية Gaia هي طريقة لهندسة النظم متعددة الأدوات تستخدم لوصف مرحلتي التحليل والتصميم بمستوى نظري متقدم[36]. تشير مرحلة التحليل إلى تحديد نموذج للأدوار والتفاعلات. أما مرحلة التصميم فهي المرحلة التي يتم فيها ترجمة هذه المفاهيم إلى نماذج أدوات وخدمات ومعلومات. يبدو أن منهجية Gaia تقتصر على النظم الصغيرة المستخدمة في الشركات الثابتة مع أن التطور الذي طرأ على Gaia يتيح نمذجة قوانين الشركة وتطبيقها على النظام.

تتضمن منهجية Tropos[10] أساليب عديدة مختلفة لتحليل وتصميم المتطلبات. فهذه المنهجية تتيح نمذجة المستخدمين والأهداف المتعلقة بالأجهزة والبرامج والخطط والموارد والعلاقات والتبعيات بين ذلك كله. يبدو أن منهجية

Tropos مصممة للنظم المغلقة بسبب نموذج هيكلية الأدوات الداخلية التي تتيح للمرء تعريفه.

في القسم 4-7، سنناقش كيف يمكن الانتقال من هندسة البرمجيات أدواتية التوجه AROSE إلى هندسة البرمجيات خدمية التوجه SOSE من خلال الاستفادة من الأساليب أدواتية التوجه الملائمة لتصميم الهيكلة خدمية التوجه.

4 ـ 3 تأثير الأدوات في الهيكليات خدمية التوجه

الهيكلية خدمية التوجه SOA هي «مجموعة من المكوّنات التي يمكن الاستعانة بها، ويمكن نشر وصف الروابط بينها واكتشافها»[4]. أما الخدمات فهي كوائن خاصة بالشبكة يمكن عنونتها وهي ذات روابط معيارية محددة جيداً. لهذه الخدمات روابط غير معلنة، وهي متاحة للمستخدمين حيث تكون في حالة خمول إلى أن يصدر طلب الاستخدام. تتواصل الخدمات عن طريق بروتوكولات معيارية ويمكن الوصول لها واستخدامها من دون الحاجة إلى عمليات دمج. قد تستخدم الخدمة بعدة حالات مختلفة، ذلك أنها غير مرتبطة بسياق محدد.

ثمة تشابه جزئي بين الفكرة العامة للخدمة والفكرة العامة للمكوّن البرمجي. فالخدمات مقترنة بشكل طفيف كما هو الحال في المكوّنات، وكلاهما مصمم بمعزل عن السياق الذي يستخدمان فيه. فهما يميزان نفسيهما عن طريق المستوى النظري في كلٍ منهما. تشتمل الهيكلية خدمية التوجه SOA على العديد من التنظيمات التي تتفاعل مع النظم المرتبطة من خلال شبكة، حيث لا يوجد مصمم واحد يضطلع بكل المعرفة والتحكم والملكية، بل هي موزعة على عدة مصممين. الخدمات هي مكافئ معنوي للسلعة، وتملكها شركة معيّنة، وهي ذات دلالة وهدف بالنسبة إلى بعض عملاء الشركة. بهذا المعنى، تختلف الخدمات بطبيعتها كثيراً عن المكوّنات.

كما هو الأمر لمعظم الحالات المعروفة من الهيكليات خدمية التوجه، ونظراً إلى علاقتها مع سيناريوهات الأعمال الإلكترونية، يمكن وصف هيكلية خدمات الويب كنظام ذاتي يوفر فيه الخادم (Server) بعض المنافع الذاتية للتوابع (Clients) (مثال على ذلك، يتم شراء الخدمة من قبل التابع).

تعرض هيكلية خدمات الشبكية المفتوحة[21] خصائص مماثلة. لكن،

ونظراً إلى ارتباطها مع الشبكية وسيناريوهات تطبيقات العلوم الإلكترونية، زاد تركيزها على الكفاءة العامة والاستفادة المثلى من الموارد مقابل السلوك ذي المصالح الذاتية.

يؤدي التنسيق والتركيب دوراً رئيساً في الهيكليات القائمة على مفهوم الخدمة. عادة ما تكون الخدمات تعاونية: فقد يتم استدعاؤها من قبل خدمات أخرى لتنفيذ مهمة معيّنة. إضافة إلى ذلك، تعتبر الهيكلة خدمية التوجه نظاماً شبكياً متعدد التنظيمات حيث لا يكون هناك مصمم واحد ذو معرفة وسيطرة وملكية كاملة. هذا ويمكن نشر الخدمات وتعديلها وإلغاؤها في أي وقت. بيد أن ذلك ينطوي على تأثير كبير نتيجة التغيير. أخيراً، تتميز الهيكلة خدمية التوجه بالانفتاح وعدم التيقن، ذلك أنه ليس من الممكن توقع جميع الاحتمالات أو وصف جميع الاستجابات المحتملة للنظام مقدماً. لمعالجة القضايا المذكورة أعلاه، من المهم الاعتماد على:

1. كائن برمجي ذو قدرة على معالجة أمور التنسيق والتركيب في نظام مفتوح يتسم بتأثر عالٍ بالتغيير.
2. مواصفات قابلية قراءة البيانات المتبادلة بين الخدمات ودلالات الخدمات آلياً في مجال الاهتمام على وجه الخصوص.

بالنسبة إلى البند الأول، تحقق عملية أتمته تنسيق وتركيب الخدمات فوائد مهمة من التقنيات المشتقة من سيناريوهات استخدام الأدوات (Agents). تم تعريف الأدوات على أنها «برامج تُشغّل بدلالات عالية بما فيه الكفاية لتشكّل ارتباطات جديدة مع برامج أخرى بهدف تنفيذ الأعمال»[22].

الأدوات هي نموذج حوسبة موجّه بالأهداف والأدوار. الأدوات فاعلة تحديداً لبناء البرمجيات للنظم المعقدة المرتبطة بالشبكات حيث لا يوجد سيطرة شاملة. أما في النظم ذات الديناميكية العالية فيتطلب الأمر بعض الأحيان استخدام الأدوات للتركيب الذي لا يعتمد على معلومات مسبقة فحسب. في بعض الأحيان، يمكن معرفة المظاهر التي تحدد خصائص الخدمة في أثناء فترة التنفيذ فقط (مثال ذلك، عملية تحميل النظام) أو قد تكون هذه المظاهر حاسمة بالنسبة إلى خدمات العملاء.

يتشابه سيناريو تركيب الأدوات الديناميكي المؤتمت لحل المشكلات

الموزعة PDS في النظم ذات الأدوات المتعددة، حيث تتوفر بعض مصادر المعرفة التي يجب أن تعثر على حلول مشتركة للمشكلة بطريقة غير مركزية. في سياق حل المشكلات الموزعة، لا يكون كلُّ مصدر من مصادر المعرفة قادراً على تحقيق الحل بشكل مستقل؛ لذا يتم تحليل المشكلة إلى مهام فرعية تخصص إلى مصادر معرفة آخرين.

اقترح ديفيس وسميث (Davis and Smith)[18] بروتوكول الشبكة التعاقدي (CNP)[31] الذي سن التفاوض على أساس طرح العطاءات لحل المشكلات الموزعة. التفاوض هو «... نقاش يتبادل فيه الأطراف ذوو العلاقة المعلومات، بحيث يتوصلون إلى اتفاق في نهاية الأمر»[18]. في أهم الحالات العامة، يكون النقاش عبارة عن عملية تتضمن أطرافاً قد يكونون بشراً أو أدوات برمجية. قامت مؤسسة الأدوات المادية الذكية (FIPA)[20] بوصف التفاعل بين الأدوات المشتركة في بروتوكول الشبكة التعاقدي عن طريق الخطوات الآتية:

1. يرسل البادئ طلب الحصول على عرض.

2. يقوم كل مشترك بعرض الطلبات المستلمة للحصول على العروض (من أطراف مختلفين) ويطرح عطاءات وفقاً لذلك.

3. يختار البادئ أفضل عطاء، ويمنح العقد للمشاركين القائمين للعطاء الأفضل ويرفض العطاءات الأخرى.

يناقش المرجع[3] أوجه التشابه بين بروتوكول الشبكة التعاقدي CNP (والمشكلة التي يعالجها) والمشكلات والحلول التي تعالجها هيكلية خدمات الويب WSA.

بالنسبة إلى البند الثاني، من الأهمية بمكان الحصول على وصف لبعض النماذج بطريقة قابلة لقراءة البيانات المتبادلة آلياً وعلى الإمكانيات التي يوفرها جهاز الخادم (Server). في هذا السياق، من الأمور الرئيسة أن يتم تحديد أي مظاهر الخدمة قد عرّفت في وصف النموذج بطريقة ملائمة (مثلاً، وظيفة الخدمة، الخصائص غير الوظيفية والسلوكية) وما هي اللغة المُستخدمة للتعبير عن هذه المظاهر. في المرجع[23]، نوقشت ثلاث طرق لوصف الخدمة: الوصف النصي (أي، يتم البحث نموذجياً عن طريق مطابقة الأنماط)، الوصف الاستنباطي (أي، يتم التعبير عن خصائص الخدمة على هيئة الربط بين الخاصية

والقيمة)، والتوصيف. أما ميزة استخدام النوع الثالث بناءً على التوصيفات فتكمن في القدرة على الاستدعاء (أي «عدم وجود العثرة أمر سلبي» (Absence of False Negative)) والدقة والإحكام (أي «عدم وجود العثرة أمر إيجابي» (Absence of False Positive)) في عملية البحث.

4 ـ 4 الهيكلية القائمة على النماذج لخدمات الأدوات الشبكية

يعود سبب التحسّن الفعلي الذي توفره الهيكلية خدمية التوجه في تطوير الوسائط إلى التوجه نحو النظم القائمة على خدمات الويب. تتيح خدمات الويب استخدام الوسيط لدعم الهيكليات خدمية التوجه مع بعض الخدمات ضعيفة التقارن التي من شأنها حل العديد من المشكلات التشغيل المتداخلة.

بطريقة مماثلة، يمكن التعامل مع النظم الشبكية كوسيط ذي توجه خدمي ناشئ باستخدام مكوّنات ضعيفة التقارن وتحويل التركيز إلى تشارك الموارد بدلاً من التركيز على التنسيق البسيط عن بعد. أنشأت الشبكية كبيئة تشغيل ثابتة حيث يتم استخدام الموارد من قبل تخمين الدفعات بما في ذلك التفاعلات المنعدمة أو النادرة[2، 21]. لقد تطور الأمر بحيث أصبحت البيئات أكثر ديناميكية حيث يمكن اكتشاف الموارد واستثمارها بعدة وسائل من وسائل الأدوات المضمنة في التطبيقات ذات الخدمات المكثفة[21].

يتضمن هذا السيناريو أن إطار العمل لدمج الوسيط يتكون من التقارب بين تقنيات الشبكية وتقنيات خدمات الويب والوسيط الحالي في الهيكلية القائمة على النموذج خدمي التوجه. في هذه العملية، يمكن استخدام الهيكلية القائمة على النموذج لوضع التقنيات مع بعضها البعض، وعرض طريقة لتصميم برمجية ذات منصة مستقلة. تقود هذه الاعتبارات إلى نمذجة ذات منصة مستقلة لخدمات الشبكية[1]. تنص مسودة حديثة لوثيقة استخدام الأداة Globus Tookkit 4 (GT4)[30] على أن «يتم بناء التطبيق خدمي التوجه من خلال تركيب المكوّنات المحددة بواجهة الخدمة (وهي خدمات الويب حسب السياق الحالي)». تجمع هذه الجملة رؤية التقارب بين مكوّنات الوسائط وخدمات الويب وخدمات الشبكية في الهيكلية خدمية التوجه.

الشبكية المفتوحة OGSA وإطار عملي موارد خدمات الويب WSRF[8] نموذجاً للنظم الشبكية يكون فيه كل شيء ممثلاً كخدمة: «كيان مخوّل من

شبكة الحاسوب يوفّر بعض الإمكانيات من خلال تبادل الرسائل»[21]. هذا النموذج هو فعلياً هيكلية ذات توجه خدمي تدعم كل من التواصل عن بعد أو محلياً لتوفير التوافقية بين المكوّنات.

في هيكلية خدمات الشبكية المفتوحة، تعرّف خدمات الشبكة بأنها خدمة ويب خاصة توفّر مجموعة من الواجهات الأساسية وتتبع تحولات محددة. أما إطار عمل موارد خدمات الويب فيحاول دمج منهجية هيكلية خدمات الشبكية المفتوحة بتقنيات الويب الدلالي. فهو يوسع آلية لغة وصف خدمات الويب WSDL ما يتيح استخدام خدمات ويب وخدمات شبكية مميزة. إن وجود حالة هو موضوع مركزي لأنها تتيح تمييز حالة طلب خدمة من حالات الطلبات الأخرى. بهذه الطريقة، يمكن عرض هيكلية خدمات الشبكية المفتوحة كنظام موزع حيث يكون لكلّ طلب خدمة هوية فريدة خاصة به، كما يكون لكلّ طلب حالة معيّنة.

إن التقارب بين الويب والشبكية المتحقق في إطار عمل موارد خدمات الويب WSRF له بعض التضمينات ذات المعنى بمستوى متقدم، تحديداً في الشبكية الدلالية[16] وتصور خدمات الويب الدلالي[35] في هيكلية الخدمات الشبكية (GSA) الموضح من حيث هيكلية خدمات الشبكية المفتوحة[21]. تحاول الشبكية الدلالية إيجاد بيئة تركز على الإنترنت حيث تتشارك الموارد ويتم إدارتها اعتماداً على دلالات الربط البيني[39]. تضيف خدمات الويب الدلالية دلالات إلى خدمات الويب مستغلةً توصيف الخدمة الموصوفة في لغة الترميز الخاصة بوكالة مشاريع أبحاث الدفاع المتقدمة DAML(*)، أو لغة توصيف مصطلحات الويب OWL[34] أو لغة النمذجة الموحدة[27]. يهدف الباحثون في مجال خدمات الويب الدلالي والشبيكة الدلالية إلى تطبيق تقنيات الويب الدلالي على خدمات الشبكة وذلك لإضافة دلالات للبرمجيات خدمية التوجه، التي تعتبر متطلباً أساسياً لتمكين الأدوات والسلوكيات المستقلة في الهيكليات خدمية التوجه (SOA). هذه الرؤية متعامدة مع التقارب بمستواه الأدنى بين الشبكة والإنترنت والمتحقق بواسطة الشبكية المفتوحة OGSA وإطار عملي موارد خدمات الويب WSRF. لدمج جميع التقنيات المبيّنة أعلاه في

(*) بدأ برنامج إعداد لغة الترميز التابع لوكالة مشاريع أبحاث الدفاع المتقدمة DARPA رسمياً عام 2000. والهدف من لغة الترميز هذه هو تطوير لغة وأدوات لتسهيل مفهوم الويب الدلالي (المترجم).

الشبكة الدلالية خدمية التوجه، يمكن استخدام الهيكلية القائمة على النماذج MDA[26] كموائم. تعتبر التحولات في النموذج إحدى أهم مزايا الهيكلية القائمة على النماذج. تستخدم مجموعة من القوانين والتقنيات لتحويل نموذج موصوف بلغة النمذجة الموحدة إلى نموذج آخر. في الهيكلية القائمة على النماذج، تمثل النماذج أجزاء من الوظائف أو بُنى أو سلوكيات النظام بدرجات مختلفة من التفاصيل. إن فصل طرق عرض النموذج المنتقاة عن طرق العرض المبسطة تسمح بإجراء التكبير ـ في العديد من النماذج البديلة لنفس وظائف النظام. توفر الهيكلية القائمة على النماذج منهجية ذات مستوى متقدم لتطوير الخدمات وتجيز إدارة دلالات ودورة حياة تطبيقات الأعمال. كما أنها تتيح إجراء تكامل مع تقنيات أخرى وصيانة التطبيقات المعقدة التي تعمل ضمن هيكلية خفيفة. إضافة إلى ما تقدم، الهيكلية القائمة على النماذج في الهيكلية خدمية التوجه تخوّل المنهجية الموجهة بالخدمات لتطوير تطبيقات الأعمال[30]. لذا، تعتبر الهيكلية القائمة على النماذج أداة مستقلة فاعلة كلغة النمذجة الموحدة، كما أنها أداة محايدة على نحو فعال للبرمجيات كائنية التوجه.

في الختام، إن التقارب بين خدمات الشبكة والإنترنت هو الخطوة التالية في تطوير هيكلية الخدمات الشبكية GSA. فمن ناحية، تبسّط خدمات الويب برمجة الشبكة، ومن ناحية أخرى فإنها تضيف بعض الآليات المفيدة جداً استقلالية المنصة، ما يسمح لنا بجمع خدمات الويب وخدمات الشبكة والأدوات معاً.

4 ـ 5 تنسيق الأدوات والتزامن في هيكلية خدمات الويب

الخدمات في جوهرها هي كيانات عديمة الحالة. إضافة إلى ذلك، تتطلب العديد من سيناريوهات أعمال ـ أعمال إلى سلوك ذي حالة محددة وتعاونية، وهذا ينطوي على تنسيق معقد. لذا فإنه يجب تحديد الترتيب بشكل مضبوط وبيان سبب الحاجة للعمليات الخدمية (على سبيل المثال: يتم الشراء بعد الدفع). في سيناريو خدمة الويب، ما من شيء يمنع تنسيق الخدمات على مستوى الشيفرة البرمجية للبرنامج. على أي حال، تركز معظم لغات التنسيق على إجراء فصل واضح بين التنسيق والحوسبة، وينظر إليها على أنها قضايا تتعلق بتصميم لغة متعامدة. ثمة استعارة شهيرة[9] على هيئة معادلة بسيطة تستخدم تؤسس لهذه التعامدية على النحو الآتي:

البرنامج = الحوسبة + التنسيق (4.1)

بناء على ذلك، وحسب هذه الاستعارة، يجب أن تتكون لغة البرمجة من مكوّنين متعامدين: مكوّن التنسيق ومكوّن الحوسبة. هذا الفصل فعّال من وجهة نظر هندسة البرمجيات: فالحفاظ على القضايا الخاصة بالتنسيق مفصولة عن القضايا المتعلقة بالحوسبة يسبب تفاعلاً على مستوى متقدم من الملخّص، ما يعني تبسيط مهام البرمجة. هذا الفصل أمر حاسم في الهيكلة خدمية التوجه: في النظام ذي التقارن الضعيف، المتميز بتأثير الكبير للتغيير والمعيارية كميزة هامة. منذ عقد التسعينيات من القرن الماضي، أمكن التعبير عن منطق العمل وإدارته بصورة مستقلة عن البرنامج بواسطة نظام إدارة سير العمل المركزي (WfMS). في سياق هذا النظام، يمكن إعادة كتابة المعادلة (4.1) بالصورة الآتية[25]

سير العمل = النشاطات + العمليات (4.2)

حيث تكون الكيانات الأساسية في مخطط سير العمل عبارة عن النشاطات التي تنفذ وحدات العمل، بينما توظف العمليات لأداء التنسيق كالنشاطات. تتبع هذه المنهجية من قبل اللغات القائمة على XML الحالية (مثال ذلك لغات التزامن ولغات التصميم) التي تحاول توحيد إمكانية تنسيق الخدمة في هيكلية خدمات الويب WSA. هناك لغات مختلفة لتحديد مظاهر الخدمة المختلفة. على سبيل المثال، تصف آلية لغة وصف خدمات الويب WSDL الواجهة، بينما تصف لغات التزامن والتصميم العملية. جميع هذه المعايير لا تلتقط دلالات الخدمة.

4 ـ 6 منهجية توصيف هيكلية خدمات الويب

يصف هذا القسم مزايا جمع المنهجية أدواتية التوجه مع هيكلية خدمات الويب في سياق ما يعرف بالويب الدلالي. توفّر خدمات الويب طريقة قياسية لتوافقية التطبيقات البرمجية التي يتم تشغيلها ضمن منصات وأطر عمل متنوعة. تم في رابطة الشبكة العالمية (W3C)[*][38] تطوير مجموعة من التقنيات التي

(*) منظمة W3C أو رابطة الشبكة العالمية (World Wide Web Consortium) هي أهم منظمة دولية لوضع المعايير لشبكة العالمية. تعمل منظمة W3C على إيجاد ووضع قواعد ومواصفات ومعايير الأساسية وتطوير الحالية. معايير W3C تقود شبكة الويب إلى الامام، وتكون عادة سابقة الأوان من الممارسات الحالية. هدفها هو تحسين التفاعل بين مستخدمي الشبكة وتوفير نماذج موحدة للناس للتعاون. تشارك W3C أيضاً في التربية والتوعية وتطوير البرمجيات وهي بمثابة منتدى مفتوح لمناقشة الأمور المتعلقة بالويب (المترجم).

قادت هيكلية خدمات الويب إلى الجهد الكامن الكامل بطريقة نجاح مثل هذه التقنيات في النمو في القطاعين الصناعي والأكاديمي ولا يمكن استثمارها بالكامل بعد. أما سبب هذا الوضع فهو عدم وجود تقنيات دلالية وبنى تحتية. يهدف الويب الدلالي إلى إتاحة مشاركة المعرفة بين المصادر الموزعة والديناميكية غير المتجانسة. تمثل الأدوات طريقة طبيعية لتنفيذ هذه الأنواع من المهام. فكما يؤكد المرجع[33]، هذه الأدوات تناسب عمليات محتويات الويب ذات الدلالة. إضافة إلى ذلك، فهي تعرض قدرات اجتماعية واستقلالية. الأداة هي الجزء الأساسي في البرمجية، تعمل على تنفيذ مهمة معيّنة، بينما الخدمة هي مورد يتميز بواسطة مجموعة تجريدية من الوظائف.

في المرجع[6]، تم شرح رؤية الويب الدلالي بهذا السيناريو، يمكن أن تنفذ الأدوات البرمجية التي تنتقل من صفحة إلى أخرى مهام معقدة للمستخدمين بسهولة.

التوصيف هو المظهر الأساسي لإدراك رؤية الويب الدلالي. من وجهة النظر هذه، التوصيف هو عبارة عن وصف شكلي للمفاهيمية. فهي تعبر عن مدى من المفاهيم (نطاق معرفي معيّن) والعلاقات بينها والقيود المنطقية التي تحكم هذا النطاق. من الممكن تعريف التوصيف في الويب بمجموعة من اللغات كإطار عمل وصف المصادر (RDF)[29] وOWL(*)[28]. في هذا السياق، يوفر التوصيف أساساً لتحديد مشكلة التقاط دلالات الخدمة. إن القدرة على فهم ما تؤديه الخدمة أمر حاسم للأداة البرمجية بهدف اكتشاف واختيار وتكوين الخدمات. بطريقة مماثلة، فمن الأهمية بمكان توفر القدرة على وصف والتقاط الأدوات التابعة لها وغيرها من الأدوات المطلوبة في أثناء فترة التنفيذ، وذلك للتغلب على المشكلات التي لم تحدد مسبقاً. أُجري العديد من الأبحاث لتحديد النقص في الدلالات. هناك العديد من الجوانب التي يجب أن تراعى في التوصيف المستخدم لحوسبة الخدمات؛ إذ يجب أن توفر آليات لتتيح الاكتشاف والتركيب والتزامن والتنسيق. على سبيل المثال، OWL-S[13] هي توصيف خدمات الويب تخص OWL. فهدف هذا التوصيف تسهيل أتمتة اكتشاف وتنفيذ وتركيب خدمات الويب وعملياتها الداخلية.

(*) لمزيد من المعلومات، انظر: <http://www.w3.org/TR/owl-features>.

تم بناء OWL-S بناءً على ثلاثة مفاهيم جوهرية هي:

- الملف، الذي يلتقط المعلومات اللازمة لاكتشاف الخدمة والإعلان عنها.
- المعرفة الأساسية، التي توفر معلومات عن بروتوكولات النقل وكيفية التفاعل مع الخدمة.
- النموذج، التي توفر وصفاً عن كيفية استخدام الخدمة.

من الإسهامات الأخرى، توصيف نمذجة خدمة الويب WSMO(17) الذي يوسع نطاق إطار عملي نمذجة خدمة الويب WSFM(19). بناءً على إطار عملي نمذجة خدمة الويب، يحدد توصيف نمذجة خدمة الويب أربعة عناصر من الأعلى إلى الأسفل، والتي يجب أن تراعى بغية وصف خدمات الويب. هذه العناصر هي: التوصيفات وخدمات الويب والأهداف والوسطاء. بتحديد هذه العناصر، يوفر توصيف نمذجة خدمة الويب تراكيب لتحديد التوصيف ووصف إمكانية خدمات الويب (على سبيل المثال، الافتراضات والشروط المسبقة والشروط اللاحقة) والبينيات (كالتنسيق والتزامن) وذلك لوصف الأهداف التي يرغب المستخدم بتحقيقها من خلال خدمة الويب، ولتحديد الوسطاء التي تتيح التعامل مع قضايا الموائمة والدمج والتحويل (على سبيل المثال، حل عدم التطابق المحتمل في العروض بين التوصيفات). في سياق الويب الدلالي، يقدم المرجع[15] آلية تصميم تتيح لنا التعامل مع تغير الخواص في التفاعل مع خدمات الويب. حقق هذا العمل آلية التصميم هذه في منصة تطوير خدمات الويب الدلالي IRS-III (14) باستخدام توصيف نمذجة خدمة الويب(17).

على الرغم من أن OWL-S وتوصيف نمذجة خدمة الويب تغطي قضايا كثيرة ذات صلة بدلالات خدمات الويب، إلا أن مثل هذه التوصيفات لا تراعي المشكلات المتعلقة بالجوانب الديناميكية. على سبيل المثال، هذه التوصيفات لا تراعي كيفية التعامل مع الأوضاع التي لم تحدد مسبقاً (أي التنسيق الديناميكي). ترتبط الفكرة الأساسية للتنسيق الديناميكي بالنظم المفتوحة حيث لا تكون عمليات ومصادر النظم معلومة في فترة التصميم. في مثل هذا الوضع، يكون من المفضل وجود آلية تتيح للعمليات (أو مجموعة مختارة منها) نقل الهدف من ورائها والحاجة إلى المصادر في أثناء فترة التنفيذ. تم اقتراح منهجية توصيف ممكنة لهذه المشكلة في المرجع[32]، حيث حدّدت توصيفاً للتنسيق. أما المفاهيم

الرئيسة فهي: الأداة والعملية والمصدر الاعتمادية والعلاقة التشغيلية. تحديداً، عندما يتم تحديد النوع الفرعي لعملية مفهوم نشاط التنسيق. وهذا المفهوم بدوره هو عبارة عن تراكيب من النشاطات، وبعضها يكون صغيراً (أي لا تتكون من نشاطات أخرى فرعية). تقارن مبادئ النشاط الصغير بمبادئ المعالجة في OWL-S. يتيح التوصيف التعبير عن عدد من مزايا النشاط التنسيقي كالموعد المبكر لبدء النشاط والفترة الزمنية المتوقعة لإنجازه والأداة التي يمكن أن تنفذ هذه النشاط. من خلال مفهوم المصدر (Resource) يمكن وصف طبيعة المصادر التي قد تكون مطلوبة لتنفيذ النشاط. على سبيل المثال، قد يكون المصدر قابلاً للاستهلاك (أي أن استخدامه يقلل من مدى توافره) وقد يكون قابلاً للمشاركة (أي أنه يمكن أن تستخدمه أكثر من أداة واحدة في أي وقت). في نموذج الاعتمادية، يتم تحديد مفهوم العلاقة التنسيقية، التي قد تكون إما تنسيقاً إيجابياً أو سلبياً. على سبيل المثال، التنسيق السلبي هو علاقة تسبب بعض الفشل إن حدثت، بينما التنسيق الإيجابي يجعل من الممكن تنفيذ نشاط آخر إن حدث. تستخدم العلاقة التشغيلية لحل العلاقات التنسيقية بين النشاطات. على سبيل المثال، إذا وجدت علاقة سلطة تعاقدية بين أداتين، فإن الأداة الأولى تأخذ أولوية أعلى من الأداة الثانية بسبب بعض القوانين المعرفة مسبقاً في السياق الذي تنتمي له.

4 ـ 7 الاستنتاجات

قمنا في هذا الفصل بإلقاء الضوء على الاهتمام المتزايد في التعامل مع كل من النظريات أدواتية التوجه والأساليب المتبعة لتنفيذها مع دعم لتطوير النظم الموزعة والمفتوحة التي توفّرها الهيكلة خدمية التوجه.

بدءاً من الأساليب أدواتية التوجه المستخدمة في تطوير البرمجيات، يمكننا أن نفترض أعمال البحث التي تسعى إلى تحديد أساليب تطوير البرمجيات خدمية التوجه. حتى نمكّن تفعيل أكثير الأساليب أدواتية التوجه نجاحاً في تصميم البرمجيات خدمية التوجه، يجب أن توفر مثل هذه الأساليب مرحلة لاستنباط كيفية ارتباط الوظائف ـ المعرفة حسب نموذج محدد للوظائف ـ مع كل خدمة. يجب أن تعطينا هذه الأساليب إمكانية تحديد العلاقات بين الوظائف التي تتضمنها الخدمات المعقدة. بهذه الطريقة، يتم بناء هذه الخدمات المعقدة فوق الخدمات الأخرى مباشرة بحيث ترتبط بالوظائف المعرفة في نموذج الوظائف. قد يكون هناك مرحلة ثانية لنشر الخدمات ذات المرتبطة بالوظائف وقد تتضمن هذه

المرحلة من اختيار مكوّنات البرمجية لربطها مع كل وظيفة. قد تكون هذه الطريقة دورية تكرارية وتتيح إمكانية تحسين الخدمات ونموذج الوظائف.

من توجهات البحث المهمة ما يختص بتجسير الهيكلية القائمة على الوحدات والشبكية وخدمات الويب الدلالية لتمكين تطوير التطبيقات في الشبكة القائمة على النموذج ذي التوجة الخدمي. لهذا الغرض، يجب أن يتم التحقق بعمق من مفهوم الخدمة حتى يتم استغلالها في تطوير البرمجيات الموزعة.

أخيراً، تعتبر أعمال الأبحاث من الأمور المهمة المعنية بتحسين مفهوم الخدمة وذلك لدمج الأدوات والخدمات والدلالات. يعتبر وجود «حالة» تعبر عن الخدمة ودلالاتها قضية مركزية تحتاج إلى المزيد من التحقق، وذلك بهدف تعريفها تعريفاً ملائماً في نموذج التوجه الخدمي.

المراجع

1. S. Andreozzi [et al.]. «Towards a metamodeling based method for representing and selecting grid services.» In M. Jeckle, R. Kowalczyk, and P. Braun (eds.). paper presented at: *Proceedings of the First International Conference on Grid Services* Engineering and Management, LCNS 3270. Berlin/Heidelberg, Germany: Springer, pp. 78- 93, September 2004.

2. R. Baxter. *A Complete History of the Grid. Software Development Group EPCC and NeSC,* October 2002.

3. L. Bocchi, P. Ciancarini, and R. Lucchi. «Atomic Commit and Negotiation in Service Oriented Computing.» Technical Report UBLCS-2005-16, University of Bologna, Italy, 2005. <ftp://ftp.cs.unibo.it/pub/techreports/2005/2005-16.pdf>.

4. D. Booth, H. Haas, and A. Brown. «Web services Glossary.» Technical Report, World Wide Web Consortium (W3C), 2004. <http://www.w3.org/TR/wsgloss/>.

5. D. Booth [et al.]. «Web service Architecture.» Technical Report, World Wide Web Consortium (W3C), 2004. <http://www.w3.org/TR/wsarch/>.

6. T. Berners-Lee, J. Hendler, and O. Lassila. «The Semantic Web.» *Scientific American*: vol. 284, no. 5, 2001, pp. 28-37.

7. J. Bloomberg. «Principles of SOA.» *Application Development Trends Magazine*: vol. 10, no. 3, March 2003, pp. 22-26.

8. K. Czsjkowski [et al.]. *The WSResource* Framework, March 2005.

9. N. Carriero and D. Gelernter. «Coordination languages and their significance.» *Communications* of the ACM: vol. 35, no. 2, 1992, pp. 97-107.

10. J. Castro, M. Kolp, and J. Mylopoulos. «Towards requirements-driven information systems engineering: The TROPOS project.» *Information Systems*: vol. 27, no. 6, 2002, pp. 365-389.

11. P. Ciancarini, R. Tolksdorf, and F. Vitali. The World Wide Web as a place for agents. In: M. Wooldridge and M. Veloso, editors. *Artificial Intelligence Today. Recent Trends and* Developments, LNAI 1600, Berlin: Springer, 1999, pp. 175-194.

12. P. Ciancarini and M. Wooldridge, editors. *First Int. Workshop on Agent Oriented Software* Engineering, LNCS 1957. Ireland: Limerick, Berlin: Springer, 2000.

13. DAML. OWL-S 1.1 Release. <http://www.daml.org/services/owls/1.1/>.

14. J. Domingue [et al.]. IRS III: «A platform and infrastructure for creating WSMO-based semantic web.» paper presented at: *Proceedings, Workshop on WSMO* Implementations (WIW), Frankfurt, Germany, 2004.

15. J. Domingue, S. Galizia, and L. Cabral. Choreography in IRS-III-coping with heterogeneous interaction patterns in web services. In Y. Gil, E. Motta, V. R. Benjamins, and M. A. Musen (eds.). «The Semantic Web: ISWC 2000» paper presented at: *Proceedings of the Fourth International Semantic* Web Conference (ISWC 2005), Galway, Ireland. LNCS 3729. Berlin: Springer, 2005, pp. 171-185.

16. D. DeRoure, N. Jennings, and N. Shadbolt. «Research Agenda for the Semantic Grid: A Future e-Science Infrastructure.» Technical Report UKeS-2002-02, National e-Science Centre, December 2001.

17. J. Domingue, D. Roman, and M. Stollberg. Web service Modeling Ontology (WSMO) -An Ontology for Semantic Web services. <http://www.w3.org/2005/04/FSWS-/Submissions/1/wsmo_position_paper.html>, 2005.

18. R. Davis and R. G. Smith. «Negotiation as a metaphor for distributed problem solving.» In: *Readings in Distributed Artificial Intelligence*, San Francisco: Morgan Kaufmann Publishers, 1988, pp. 333-356.

19. D. Fensel and C. Bussler. «The Web service Modeling Framework WSMF.» *Electronic* Commerce Research and Applications: vol. 1, no. 2, 2002, pp. 113-137.

20. FIPA. *FIPA Contract Net Interaction Protocol Specification.* FIPA, 2001. < http://www.fipa.org/specs/fipa00029/ >.

21. I. Foster, C. Kesselman, and S. Tuecke. «The Anatomy of the Grid: Enabling Scalable Virtual Organizations.» *International Journal of Supercomputer Applications*: vol. 15, no. 3, 2001, pp. 200-222.

22. Y. Gill, E. Motta, V. R. Benjamins, and M. A. Musen. editors. «The semantic Web: ISWC 2000.» paper presented at: *Proceedings of the Fourth International Semantic Web Conference (ISWC 2005)*, Galway, Ireland. LNCS 3729. Springer, Berlin, 2005.

23. J. Hendler. «Agents and the Semantic Web.» *IEEE Intelligent Systems*: vol. 16, no. 2, 2001, pp. 30-37.

24. M. Klein and A. Bernstein. «Searching for services on the semantic web using process ontologies.» In I. Cruz, S. Decker, J. Euzenat, and D. McGuinness, editors. paper presented at: *Proceedings of the First* Semantic Web Working Symposium (SWWS'01), July 2001, pp. 431-446.

25. R. Klepper and W. O. Jones. *Outsourcing Information Technology, Systems and Services.* Englewood Cliffs, NJ: Prentice-Hall, 1998.

26. F. Leymann and D. Roller. *Production Workflow: Concepts and Techniques.* Englewood Cliffs, NJ: Prentice-Hall PTR, 2000.

27. Object Management Group (OMG). Model Driven Architecture (MDA) architecture board, July 2001.

28. Object Management Group (OMG). Omg unified modeling language specification v. 1.5, March 2003.

29. OWL Web Ontology Language Family of Specifications. < http://www.w3.org/2004/OWL >, 2004.

30. Resource Description Framework (RDF). < http://www.w3.org/RDF >.

31. R. Radhakrishnan and M. Wookey. *Model Driven Architecture Enabling Service Oriented* Architectures, March 2004.

32. R. G. Smith. «The Contract Net Protocol: High-Level Communication and Control in a distributed problem solver.» In: *Readings in Distributed Artificial Intelligence.* San Francisco: Morgan Kaufmann Publishers, 1988, pp. 357-366.

33. V. A. M. Tamma, C. Aart, T. Moyaux, S. Paurobally, B. Lithgow Smith, and M. Wooldridge. An ontological framework for dynamic coordination. In Y. Gil, E. Motta, V. R. Benjamins, and M. A. Musen, editors.

The semantic Web-ISWC 2000: Proceedings of the Fourth International Semantic Web Conference (ISWC 2005), Galway, Ireland. LCNS 3729. Springer, Berlin, 2005, pages 638-652.

34. V. A. M. Tamma [et al.]. «Introducing autonomic behaviour in semantic web agents.» In Y. Gil, E. Motta, V. R. Benjamins, and M. A. Musen, editors. The Semantic Web: ISWC 2000. paper presented at: *Proceedings of the Fourth International Semantic* Web Conference (ISWC 2005), Galway, Ireland. LCNS 3729. Berlin: Springer, 2005, pp. 653-667.

35. World Wide Web Consortium (W3C). OWL. Web Ontology Language, W3C recommendation, February 2004.

36. World Wide Web Consortium (W3C). Web services Semantics (WSDL-S): W3C member submission, November 2005.

37. M. Wooldridge, N. Jennings, and D. Kinny. «The Gaia methodology for agent-oriented analysis and design.» *Autonomous Agents and Multi-Agent Systems*: vol. 3, no. 3, 2000, pp. 285-312.

38. M. Wooldridge. «Agent-based software engineering.» *IEE Proc Software Engineering*: vol. 144, no. 1, 1997, pp. 26-37.

39. W3C Web services Activity. <http://www.w3.org/2002/ws/>.

40. H. Zhuge. Semantic Grid: Scientific Issues, Infrastructure, and Methodology. *Commun. ACM*: vol. 48, no. 4, 2005, pp. 117-119.

41. F. Zambonelli, N. R. Jennings, and M. Wooldridge. «Developing Multiagent Systems: The Gaia methodology.» *ACM Transactions on Software Engineering and Methodology*: vol. 12, no. 3, 2003, pp. 317-370.

5

اختبار البرمجيات كائنية التوجه

ليوناردو مارياني (Leonardo Mariani)
ماريو بيتسي (Mauro Pezzè)

5 ـ 1 المقدمة

عملية تطوير البرمجيات هي عملية معقدة وعرضة للأخطاء، وقد تفشل في تحقيق أهداف الجودة إذا لم يتم التحقق من الخصائص المطلوبة بالطريقة الملائمة. لا يمكن إضافة الجودة كفكرة متأخرة، لكن يجب أن يتم العمل على تحسينها في أثناء عملية تطوير المنتج النهائي والتحقق منه. الاختبار والتحليل هما عنصرا عمليات تطوير البرمجيات التي تهدف إلى تحديد الأخطاء التي تحدث في أثناء التطوير، كما تهدف إلى قياس مدى استعداد المنتج النهائي للاستخدام. تترجم عملية تطوير البرمجية احتياجات المستخدمين وتقييدها في مواصفات المتطلبات، كما إنها تحوّل المتطلبات إلى مواصفات الهيكلية والتصميم لإنتاج شيفرة برمجية يمكن أن تحدد احتياجات المستخدم. في كل خطوة، يعمل المطوّرون على تشكيل الحلول المطلوبة عن طريق إضافة تفاصيل والاختيار من بين العديد من بدائل التصميم، في حين يعمل خبراء الجودة على التحقق من اتساق النماذج ومتعلقاتها وتوافقها مع الأفكار النظرية السابقة ومع احتياجات المستخدم.

تحدد منهجيات تطوير البرمجيات العوامل المشتركة بين عمليات تحديد المواصفات والتصميم والتحقق، كما تعمل على تقرير النماذج التي سيتم إنتاجها في أثناء عملية التطوير وتقيّد خيارات أنشطة الاختبار والتحليل.

تم تطوير العديد من تقنيات الاختبار والتحليل في سياق المنهجيات الكلاسيكية وأساليب البرمجة، التي تفترض استخدام النماذج الإجرائية في البرمجية، بمعنى أنها تتعامل مع البرامج على أنها تحويل وظيفي من مدخلات إلى مخرجات. مثلاً، تعمل أساليب الاختبار الوظيفي المألوفة على العلاقة بين المدخلات والمخرجات كأسلوب تقسيم الفئات[31]، أو الاختبار المرتكز على الفهرس[27]، في حين تتخذ اختبارات تدفق البيانات والتحكم بها[13] أسلوب البرمجة الإجرائية.

يتميز التصميم كائني التوجه من خلال سلوكه المعتمد على حالته والتضمين (Encapsulation) والتوارث وتعدد الأشكال والربط الديناميكي ويستغل التجانس والاستثناءات بشكل مكثف[28]. تعدّل هذه المزايا النماذج الإجرائية الكلاسيكية وتخفف من وطأة بعض مشكلات التصميم والتنفيذ المعروفة، لكنها تقدم نقاط ضعف جديدة تستدعي استخدام تقنيات جديدة للاختبار والتحليل. مثلاً، الأنواع (Classes) والكوائن تدفع إلى استخدام التضمين وإخفاء المعلومات، ما يقلل من العديد من المشكلات المعروفة المشتقة من الاستخدام المكثف للمعلومات غير المحلية في البرامج الإجرائية الكلاسيكية، في حين تؤجل عمليات التوارث وتعدد الأشكال من ربط الكوائن في أثناء فترة التشغيل، وقد تؤدي إلى قصور تعتمد على الربط الديناميكي بين الكوائن. لا تزال الأساليب الكلاسيكية مفيدة عند المستويات النظرية، عند التعامل مع المتطلبات المعبّر عنها بصورة مستقلة عن قرارات التصميم. لكن، لابدّ من إلحاق ذلك بتقنيات جديدة للتغلب على المشكلات الجديدة التي تنشأ عند استخدام مزايا التصميم كائني التوجه.

5 ـ 2 تأثير التصميم كائني التوجه في الاختبار

تؤثر المزايا كائنية التوجه في الاختبار والتحليل بطرق مختلفة. تتميز الأنواع والكوائن من خلال حالتها، ولا تعتمد نتائج تنفيذ العمليات على قيم العوامل فحسب، كما في البرمجيات الإجرائية، لكنها تعتمد على حالة الكوائن. على سبيل المثال، إن تأثير استدعاء العملية commit للنوع cart المبين في الشكل 5-1 لا يعتمد على قيمة معامل المستودع warehouse فحسب، بل يعتمد أيضاً على المحتويات الحالية للعربة، أي على حالة العربة. إن معظم منهجيات الاختبار الكلاسيكية تعتمد على العلاقات بين المدخلات

والمخرجات، ولا تأخذ في الاعتبار حالة البرنامج. على سبيل المثال، يتم اشتقاق سيناريوهات الاختبار لقيم مختلفة للعامل **warehouse** للعملية **commit**، لكن لا تراعى المحتويات المختلفة للعربة، وهذا يؤدي إلى التخلص من بعض المشكلات المحتملة التي تعتمد على حالة الكائن. عند التعامل مع برمجية كائنية التوجه، علينا أن نراعي العمليات وتكاملها كما لو كانت إجراءات، بل يجب أن نراعي الأنواع والكوائن أيضاً، كما أن علينا توسيع مجموعة التقنيات التي يمكن أن تتعامل مع معلومات حالة الكائن بفعالية.

تصدر الأنواع والكوائن جزءاً من حالتها وسلوكها فقط، بينما يتم إخفاء تفاصيل التنفيذ عن الكوائن الخارجية. على سبيل المثال، النوع **Cart** المبيّن في الشكل 5-1 يخفي تفاصيل الحقول **items** وتفاصيل **numTotItems** التي لا يمكن لكائن خارجي التوصل إليها. إن التقنيات الكلاسيكية لإنشاء الأساسات والأجوبة الشافية تفترض وجود رؤية كاملة للشيفرة البرمجية. هذا ويمكن التخفيف من وطأة مشكلة الأساسات بكسر التضمين عن طريق تصدير المعلومات المخفية، لكن ذلك لن يحل المشكلة. عند التعامل مع البرمجية كائنية التوجه، نحتاج إلى منهجيات جديدة تتعامل مع المعلومات المخفية بأمان.

يمكن تحديد الأنواع عن طريق تخصيص أنواع أخرى. ترث الأنواع الفرعية خصائص الأنواع الأصلية وتضيف إليها مزايا أو تعدلها. يمكننا على سبيل المثال إنشاء النوع **SecureCart** عن طريق تخصيص النوع **Cart** المبين في الشكل 5 ـ 1. فالنوع **SecureCart** سيرث خصائص النوع **Cart**، كما سيضيف بعض العمليات التي تتعامل مع صلاحيات تفويض المستخدم. إن إعادة الاستخدام المكثف للعمليات الموروثة يثير تساؤلات جديدة عن إعادة استخدام سيناريوهات الاختبار والتنفيذ الأمثل لها. العمليات (**Methods**) المشتركة بين الأنواع الأصلية والأنواع الفرعية من دون وجود تعديلات مباشرة أو غير مباشرة يمكن أن يتم اختبارها مرة واحدة فقط، لكن يجب أن تكون تقنيات الاختبار قادرة على تحديد التفاعلات غير المباشرة وتجنب المشكلات التي قد تنتج من تفاعلات غير متوقعة لم يتم اختبارها بطريقة ملائمة.

في البرامج كائنية التوجه، يمكن أن تغيّر المتغيرات نمطها ديناميكياً. نطاق التغييرات مقيّد بنمط أساسي ثابت ومعلن يعمل على ربط الأصناف الفعلية بحيث تصبح نمطاً فرعياً تابعاً له. على سبيل المثال، المتغير **myCart**

المصرح عنه للنمط Cart يمكن أن يرتبط ديناميكياً بالكوائن التي نمطها Cart بالإضافة إلى الكوائن ذات الأنماط الأخرى التي تخصص النمط Cart (على سبيل المثال، SecureCart). يتبع الربط الديناميكي التسلسل الهرمي للتوارث، لكن لا تتسلقه: فالمتغيرات من النمط SecureCart لا يمكن أن ترتبط بالكوائن من النمط Cart. بما أن المتغيرات يمكن أن تبدل نمطها ديناميكياً، فإنه يمكن ربط استدعاءات العملية في أثناء فترة التنفيذ فقط. عليك أن تراعي أن جميع حالات الربط الممكنة لكل استدعاء متعدد الأشكال يصبح غير عملي بسرعة، بما أن عدد مجموعات الربط قد ينمو بصورة أُسِّيَّة. بناء على ذلك، علينا أن نختبر التقنيات التي تختار المجموعات الجزئية المناسبة لحالات الربط الممكنة.

إضافة إلى خصائصها المميزة، تستفيد البرامج كائنية التوجه استفادة واسعة من الخصائص الإضافية التي لا تستغل جيداً في التطبيقات الإجرائية: الشمولية والاستثناءات.

معظم لغات البرمجة كائنية التوجه توفر أنواعاً عامةً (أي أنواع منفّذة بأنماط رمزية) ترتبط بأنماط أساسية عندما تكون الكوائن ذات الأنواع العامة متشابهة. على سبيل المثال، النوع hashtable في الشكل 5-1 هو حالة من النوع العام Hashtable الذي يعمل بمعاملين يمثلان المفتاح والقيمة اللذين يجب أن يكونا متماثلين مع الأنماط الأساسية عند استخدامهما (مثلاً، String وInteger في الشكل 5-1). يجب أن يراعى جميع التماثلات الممكنة عند اختبار الأنواع العامة وهذه قد تكون عديدة جداً.

أما لغات البرمجية كائنية التوجه الحديثة فتوفر بناءات صريحة لمعالجة الحالات الاستثنائية والخاطئة. على سبيل المثال، في لغة الجافا، تستخدم Java {...}try{...}catch (...) {...} finally للتحكم بحجم التعليمات التي قد تنتج استثناءات ـ وتحدد معالج الاستثناءات المحلي. تضيف معالجات الاستثناءات تحكماً ضمنياً، ويمكن تفعيلها بصورة صريحة أو ضمنية في أثناء فترة التنفيذ، على سبيل المثال، يمكن تفعيلها في حالات القسمة على صفر. مبدأ ذلك أن تقنيات الاختبار يمكن أن تأخذ في الاعتبار جميع التنفيذات الضمنية والخاطئة عند إنشاء سيناريوهات الاختبار. على كل، ينمو عدد التنفيذات الممكنة أُسّياً حتى بالنسبة إلى البرامج البسيطة.

```
1:  public class Cart {
2:          private Hashtable<String, Integer> items;
2:          private int numTotItems;
3:
4:          public Cart() {
5:                  items = new Hashtable<String, Integer>();
6:                  numTotItems = 0;
7:          }
8:
9:          public void addItem(String itemId, Integer quantity) {
10:                 items.put(itemId, quantity);
11:                 numTotItems+=quantity;
12:         }
13:
14:         public void removeItem(String itemId) {
15:                 Integer qt = getQuantity(itemId);
16:                 if (qt != null) {
17:                   numTotItems -= qt;
18:                   items.remove(itemId);
19:                 }
20:         }
21:
22:         public void updateItem(String itemId, Integer quantity) {
23:                 if (getQuantity(itemId) != null) {
24:                   removeItem(itemId);
25:                   addItem(itemId, quantity);
26:                 }
27:         }
28:
29:         public Integer getQuantity(String itemId) { return items.get(itemId); }
30:
31:         public int getNumTotItems() { return numTotItems; }
32:
33:         public boolean commit(Warehouse warehouse) {
34:                 Enumeration<String> itemIds = items.keys();
35:                 if (!itemIds.hasMoreElements()) return true;
36:
37:                 warehouse.beginTransaction();
38:
39:                 while(itemIds.hasMoreElements()) {
40:                       String id = itemIds.nextElement();
41:                       if (!warehouse.isAvailable(id, items.get(id))) {
42:                         warehouse.abortTransaction();
43:                         return false;
44:                       }
45:                 }
46:
47:                 itemIds = items.keys();
48:
49:                 while(itemIds.hasMoreElements()) {
50:                       String id = itemIds.nextElement();
51:                       warehouse.remove(id, items.get(id));
52:                 }
53:
54:                 items = new Hashtable<String, Integer>();
55:                 numTotItems = 0;
56:
57:                 warehouse.commitTransaction();
58:                 return true;
59:         }
60: }
```

الشكل (5 ـ 1): مقتطفات من مثال العربة Cart البسيط بلغة البرمجة Java

5 ـ 3 تقنيات الاختبار القائمة على المواصفات

تركز أهم تقنيات الاختبار القائمة على المواصفات المتوفرة حتى الآن للبرمجيات كائنية التوجه إما على مواصفات بيانية [تحديداً بلغة النمذجة الموحدة UML][29] أو على مواصفات أساسية [تحديداً، مواصفات جبرية][16].

توفّر لغة النمذجة الموحدة العديد من لغات التوصيف والنمذجة التي تلتقط طرق عرض وخلاصات مختلفة في أثناء مراحل تحليل وتصميم المتطلبات المختلفة. حتى الآن، ركزت الأبحاث التي تُجرى على اختبارات النوع الداخلي بشكل أساسي على رسوم بيانية للحالة تحدد سلوك النوع المعتمد على حالته، في حين راعت الأبحاث التي أجريت على اختبار النوع الداخلي النماذج العديدة كالتسلسل والتعاون ومخططات النوع البيانية.

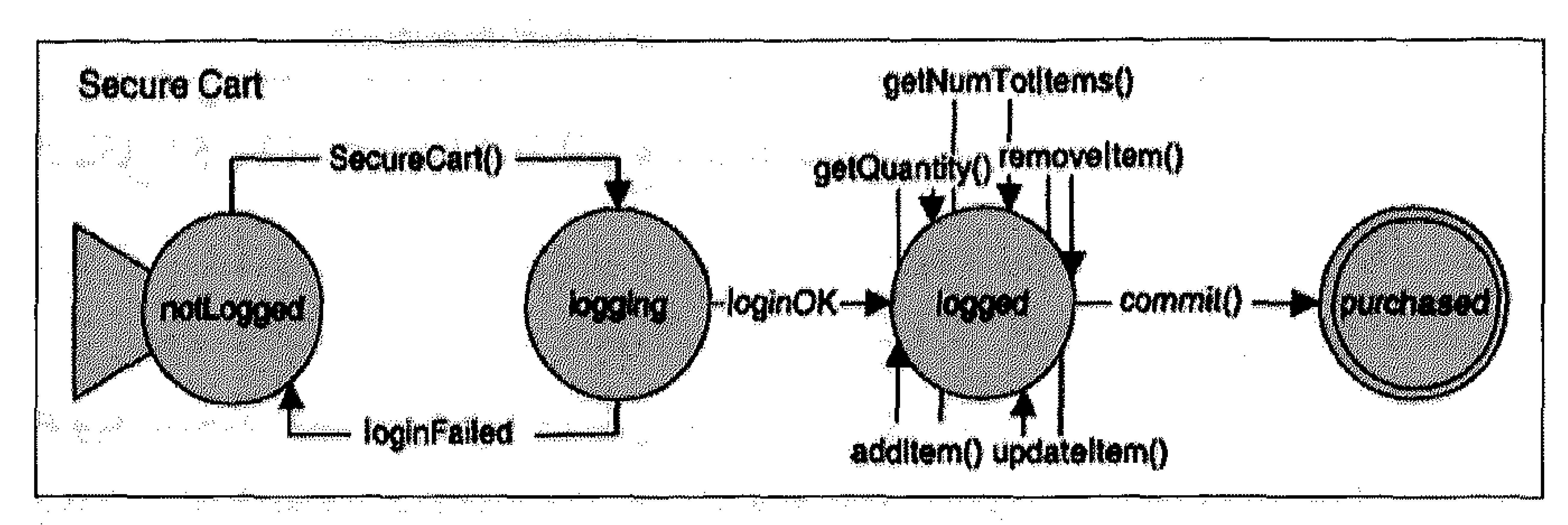
تصف المواصفات الجبرية شارات ودلالات العملية، وتستخدم لإنشاء سيناريوهات الاختبار أوتوماتيكياً. إن التقنية المعروفة باسم السيناريوهات المرادفة لا تغير من عمليات التضمين وإخفاء المعلومات، ويمكن تكييفها لتُوائم أنواع المواصفات الأخرى.

5 ـ 4 اختبار النوع الداخلي بلغة النمذجة الموحدة UML

تصف مواصفات لغة النمذجة الموحدة (UML) سلوك الأنواع القائمة على الحالة باستخدام رسوم بيانية للحالة، التي تستخدم أحياناً كآلات بيان الحالة محدودة وبسيطة (FSMs). تتكون آلة بيان الحالة المحدودة من مجموعة من الحالات ومجموعة من التحولات بين الحالات. تمثل الحالات مجموعة من قيم الخصائص التي ينتج منها ردود الأفعال نفسها التي تنتج من المحفزات التي تنتج من أيّ مراقب خارجي، بينما تمثل التحولات الأحداث التي يمكن أن تغيِّر الحالة. تعنون التحولات باسم الحدث المرتبط باستدعاءات العملية أو المرتبط بالأعمال الداخلية. في الحالة الأولى، يحدث التحول عندما يتم تنفيذ العملية، بينما في الحالة الثانية، يحدث التحول عند تنفيذ العمل الداخلي.

تميّز آلات بيان الحالة المحدودة بواسطة الحالة الأولية التي تتوافق مع حالة الكائن الأولية ومجموعة من الحالات النهائية التي تمثل مجموعة الحالات التي يمكن عندها إنهاء التنفيذ. يدخل الكائن الجديد في حالته الأولية ويتطور حسب السلوك المحدد له مسبقاً إلى أن يصل إلى الحالة النهائية.

يبيّن الشكل 5 ـ 2 مثالاً على آلة بيان الحالة محدودة تحدد عربة آمنة مضافة على تطبيق العربة الممثلة في الشكل 5 ـ 1. نحن نفترض أن دلالات التحوّل المعرّفة ضمنياً تنظم في حلقات ضمنية ذاتية[4] ـ أي أن النتائج غير المتوافقة مع أيّ من التحولات الصريحة تخرج من حالة معطاة. على سبيل المثال، يتوافق استدعاء العملية addItem () في حالة notLogged مع حلقة ذاتية، أي إن عملية الاستدعاء لا تعدل الحالة. يقدم كلٌّ من لي (Lee) وياناكاكيس (Yannakakis)[26] (Yannakakis)[26] التفاصيل الأساسية لآلات تحديد الحالة المحدودة.



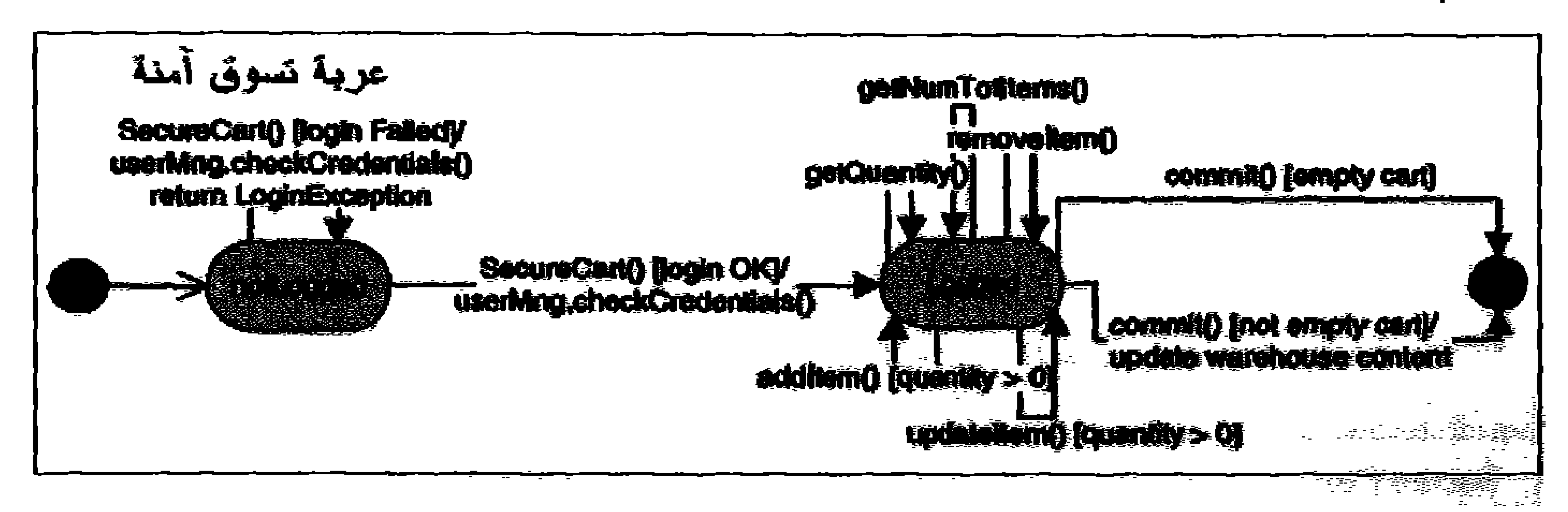
الشكل (5 ـ 2): بيان الحالة محدود وبسيط للنوع SecureCart

تفسير: الحالة الأولية مبيّنة بالمثلث، في حين أن الحالات النهائية مبيّنة بالدائرة المزدوجة. التحولات ذات العناوين المنتهية بـ () تشير إلى استدعاء عملية، والعناوين العادية تشير إلى عمليات داخلية.

توسع النوع Statecharts آلات تحديد الحالة المحدودة عن طريق زيادة عناوين التحولات لتمثيل الآثار الجانبية والحاميات (guards) وعن طريق تقديم آليات تركيب لنموذجة الحالات والتحولات بطريقة مدمجة. تكون الآثار الجانبية عبارة عن أعمال منفذة عند تفعيل التحولات ـ على سبيل المثال، تنفيذ عمليات الكوائن الخارجية أو تحديث المتغيرات المحلية. يعنون تحوّل Statechart بحدث مفرد وتسلسل فارغ من الأعمال. تفصل أسماء الأحداث عن الأعمال باستخدام العلامة (/). أما الحاميات فهي شروط مرتبطة بالتحولات. يمكن أن يتم تفعيل التحول إذا كانت قيمة الحامية خاصته «صحيح true». يتم تحديد الحاميات بعد اسم الحدث من دون استخدام الأقواس المربعة.

يبيّن الشكل 5 ـ 3 مواصفة مخطط الحالة Statechart الخاص ب SecureCart. تكون هذه المواصفة أكثر دقة من مواصفة آلة تحديد الحالة المحدودة المبيّنة في الشكل 5 ـ 2. بما إن مخطط الحالة Statechart يبيّن أن SecureCart يسبب تفاعلاً مع userMng، فإن الحالة المستهدفة تعتمد على

نتائج محاولات الدخول للنظام؛ ويمكن تنفيذ العمليتين addItem وupdateItem إذا تم تعيين قيمة موجبة لكمية المتغير فقط.



الشكل (5 ـ 3): مخطط الحالة للنوع SecureCart

تحدّد آليات التركيب الحالات المركبة (أي الحالات التي تم الحصول عليها باتحاد عدة حالات) والبنى المتزامنة (أي الأجزاء الفرعية من الحالة التي يمكن أن تتطور بشكل متزامن). قام هاريل ديفيد David Harel بوصف مخططات الحالة Statecharts بالتفصيل في بحثه[17].

من الطرق البسيطة لإنشاء سيناريوهات الاختبار من آلة تحديد الحالة FSM ومخططات الحالة Statecharts أن يتم استدعاء جميع حالات تسلسل التحول الممكنة. هذا المعيار الذي يتوافق مع تغطية المسار في اختبارات الهيكلية يقود إلى العديد من سيناريوهات الاختبار بشكل لا نهائي وبسرعة. تحدد تقنيات الاختبار العملية مجموعات صغيرة من سيناريوهات الاختبار، لكنها تكون وثيقة الصلة ببعضها البعض، وذلك بالرجوع إلى الحالات والتحولات والمجموعات المترابطة من الحالات والتحولات[26, 15, 4]. أما المعايير الأكثر شيوعاً بالنسبة إلى آلة تحديد الحالة فهي شمول الحالات وشمول التحولات وشمول حلقة الحدود الداخلية. يتطلب شمول الحالة أن يتم اجتياز جميع الحالات مرة واحدة على الأقل. إن اجتياز جميع الحالات من دون اعتبار الأحداث التي يمكن أن تتطرأ على الحالة يعني اختيار مجموعات اختبار صغيرة، لكنها نادراً ما تكون كافية لأداء اختبار متمكّن. على سبيل المثال، في حالة الاختبار الوحيدة الآتية:

(TC1) SecureCart (), loginOK, commit ()

يتم استعراض جميع حالات آلة تحديد الحالة FSM في الشكل 5-2، لكن

لا يتم اختبار العديد من العمليات، وبناء على ذلك، لا يمكن الكشف عن الأخطاء التي قد تكون موجودة في هذه العمليات.

يتطلب شمول الانتقال من حالة إلى أخرى أن يتم اجتياز جميع الانتقالات مرة واحدة على الأقل، بما في ذلك الانتقالات الضمنية، إذا كانت الدلالات المأخوذة في الاعتبار تتطلب ذلك. شمول الانتقال بين الحالات يتضمن شمول الحالة؛ وهذا يعني ضمان شمول جميع الحالات أيضاً. على سبيل المثال، حالة الاختبار الوحيدة TC1 الكافية لضمان شمول الحالة، ليس من ضمان لشمول الانتقال بين الحالات في آلة تحديد الحالة في الشكل 5-2. نحتاج هنا إلى سيناريوهات اختبار إضافية، كما في المثال الآتي[1]:

(TC2) SecureCart (), loginOK, addItem (), removeItem (),
(commit ()

(TC3) SecureCart (), loginFailed, addItem (),
(getQuantity (), updateItem (), removeItem (),
(getNumTotItems (), SecureCart (), loginOK,
(addItem (), updateItem (), commit ()

(TC4) SecureCart (), loginOK, addItem (), getQuantity (),
getNumTotItems (), commit ()

إن تغطية الانتقال من شأنه أن يحسّن من تغطية الحالة، لكنه قد يغفل عن السلوكيات غير الصحيحة التي تعتمد على عمليات التنفيذ المتكررة.

يتطلب تغطية حلقة الحدود الداخلية وجود مسارات بسيطة لاستعمالها. والمسار البسيط هو المسار الذي يبدأ من حالة أولية وينتهي بحالة نهائية ويجتاز جميع الحالات مرة واحدة فقط. أما الحلقة فهي المسار الذي يبدأ وينتهي بنفس الحالة. يمكن أن تجتاز الحلقة عدة مرات في عملية تنفيذ واحدة. أما تغطية حلقة الحدود الداخلية فيتطلب الأمر اجتياز المسار البسيط مرة واحدة على الأقل، لكن يتطلب اجتياز كل حلقة عدداً من المرات مساوياً للحدّ الأدنى والحد الأعلى. كما يتطلب ذلك اجتياز الحلقة عدداً من المرات يتراوح بين

(1) طالما أن تسجيل الحالة يمثل النتائج الممكنة فقط لمحاولات الدخول، لا نأخذ في الحسبان الانتقالات الضمنية لهذه الحالة.

الحد الأدنى والحد الأعلى. يضمن تغطية مسار الحدود الداخلية تغطية أفضل من المعايير السابقة بنفس تكلفة مجموعات الاختبار الأكبر حجماً.

إذا أخذنا في الاعتبار آلة تحديد الحالة FSM في الشكل 5-2 وقيّدنا عدد مرات تكرار الحلقة إلى 3، فإننا نحقق تغطية حلقة الحدود الداخلية عن طريق إضافة السيناريوهات التالية إلى سيناريوهات الاختبار من TC1 إلى TC4:

(TC5) SecureCart (), loginOK, addItem (), addItem (), addItem (), commit ()

(TC6) SecureCart (), loginFailed, SecureCart (), loginFailed, SecureCart (), loginFailed,
SecureCart (), loginOK, addItem (), getQuantity (), getQuantity (), getQuantity (), commit ()

قام العديد من الباحثين بتحديد معايير إضافية لمخططات الحالة Statecharts وبعض المتغيرات الأخرى الخاصة بآلة تحديد الحالة FSM كموانئ الإدخال والإخراج ذات التشغيل الذاتي I/O automata[6, 3]. وهناك طريقة مهمة على وجه الخصوص وهي Wp التي تُنتج سيناريوهات اختبار بتوفر معلومات عن السلوك المراقب[10]. تزيد هذه الطريقة من تغطية التحول عن طريق اعتماد مفهوم متسلسلات التمييز وتطلُّب تغطية متسلسلة تمييز واحدة على الأقل لكل حالة. أما متسلسلة التمييز للوضع فهي تسلسل يعمل على تمييز حالة معيّنة من غيرها من الحالات في ميناء إدخال/ إخراج ذاتي التشغيل عن طريق إنتاج مخرج واضح ومميز. إن تغطية الانتقال من حالة إلى أخرى يضمن تغطية جميع الأحداث لجميع الحالات. أما تغطية تسلسل التمييز فيضمن مراقبة التأثيرات المختلفة من المخرجات الخارجية. يتم الحصول على سيناريوهات الاختبار عن طريق وضع التسلسلات مع بعضها البعض، بتسلسل يضمن تغطية الانتقال بمتسلسلات التمييز. مثلاً، يمكن الحصول على متسلسلات التمييز لمخطط الحالة Statechart الموضح في الشكل 5 ـ 3 عن طريق استغلال المخرجات والإشارات الصادرة عن العمليتين SecureCart وcommit (). هذا وقد تم شرح متغيرات وتفاصيل الطريقة Wp في المرجعين[10] و[15].

إذا تضمن مخطط الحالة Statechart أوضاع حماية، يمكننا أن نطبق معايير تغطية مشروطة ومتفرعة على تلك الحمايات. على سبيل المثال، يمكننا

أن نطبق معيار القرار المشروط المعدل (MC/DC) عن طريق جعل كل فقرة في الحمايات تقييم حالات الصح والخطأ في أثناء تحديد النتيجة النهائية للتعبير كاملاً[30]. إن منطقية هذا المعيار هي أن كل فقرة يجب أن تختبر على حِدَة، وذلك لتجنب تأثيرات الحجب بين الفقرات المختلفة. يتضمن هذا المعيار سيناريوهات الاختبار التي تنتهك شروط الحماية. وهذه الحالات مفيدة للتحقق من كيفية استجابة الكائن المستهدف للمحفزات غير المقبولة بسبب قيم بيانات معيّنة.

على سبيل المثال، يتطلب المعيار المطبق على مخطط الحالة Statechart في الشكل 5 ـ 3 سيناريوهات اختبار تعمل على تنفيذ (أ) العملية secureCart () عند نجاح أو فشل عملية الدخول، و(ب) العملية commit () عندما تكون العربة فارغة أو غير فارغة، و(ج) العمليات addItem () وupdateItem () مع الكميات الموجبة وغير الموجبة. إذا تم تحديد الحمايات بلغة قيود الكائن OCL فإنه يمكن إنتاج سيناريوهات الاختبار بطريقة شبه آلية[5].

يمكن التعبير عن دلالات آليات تكوين مخطط الحالة من حيث مخططات حالة Statecharts بسيطة، وبذا يمكن اشتقاق سيناريوهات الاختبار بالرجوع إلى مخططات الحالة الممهدة كما ورد في Binder[3]. تسبب عملية التمهيد تحلل التسلسل الهرمي والبنى المتزامنة إلى مخططات حالة كبيرة ممهدة تمثل كافة السلوكيات الممكنة.

تتضمن العربة في الشكل 5 ـ 1 خللاً، ذلك أنه إذا استدعيت العملية addItem لإضافة عنصر موجود مسبقاً في العربة، فإنه سيتم تجاوز الكمية المخزنة في hashtable، بينما تزيد قيمة متغير الحالة numTotItems، وبذا تصبح العناصر الموجودة في العربة والكمية المسجلة في numTotITems مختلفة. على سبيل المثال، إن استدعاء العمليتين addItem ("Java Book",1) وaddItem ("Java Book",2) وتنفيذهما على عربة فارغة ينتج منه عربة تحتوي على نسختين من "Java Book" بينما يسجل numTotItems ثلاث نسخ. هذا، ويمكن كشف هذا الخلل بتنفيذ العملية addItem مرتين على الأقل بنفس العنصر، ومن ثم تنفيذ العملية getNumTotItem (). قد تتحقق تغطية الحالة والانتقال بمجموعات اختبار لا تتضمن إعادة استدعاء العملية، وبهذا لا يمكن كشف الخلل، بينما تكون احتمالية كشف الخلل أكبر في تغطية حلقة الحدود

الداخلية لأنها تتطلب وجود سيناريوهات اختبار تتضمن تنفيذ العملية addItem عدة مرات.

5 ـ 5 اختبار النوع الداخلي في لغة النمذجة الموحدة

تستخدم مخططات الحالة (Statecharts) بشكل أساسي لوصف سلوك النوع الداخلي، لكن تنفيذ الأحداث والعملية المتعلقة بالانتقال بين الحالات يوفر معلومات عن النوع الداخلي أيضاً. توفّر لغة النمذجة الموحدة بعض اللغات الأخرى لوصف العلاقات في النوع الداخلي: فالمخططات البيانية للنوع والكائن تصف بنية النظام بشكل ثابت، بينما تصف مخططات التسلسل والتشارك التفاعلات الديناميكية. هذا ولم تكن مخططات لغة النمذجة الموحدة كسيناريوهات الاستخدام والحزم ومخططات الأنشطة محط اهتمام الباحثين من حيث اختبارها.

لاشتقاق سيناريوهات اختبار النوع الداخلي من مخططات الحالة (Statecharts)، علينا أن نفسر التعليقات الخاصة بعمليات الانتقال وتحديد دلالاتها. لقد صيغت عمليات تنفيذ الأحداث والعمليات التي تعنون الانتقالات في Statecharts بطرق عديدة. فالإبلاغ عن مخطط الحالة يحدد العناوين باستخدام عمليات الاتصال المتسلسلة CSP[23]. تعطى العناوين كمتسلسلات لإجراءات الاتصال على النحو الآتي:

channel?inVal_ تمثل عملية إدخال متزامن؛ المتغير channel هو اسم القناة التي تدعم الاتصال، أما المتغير inVal فهو اسم رمزي للمُدخل المستلم من القناة.

channel!outVal_^ يمثل عملية إخراج متزامن؛ المتغير channel هو اسم القناة التي تدعم الاتصال، أما المتغير outVal فهو اسم رمزي للمُخرج المرسل من القناة.

تكون عمليات الإدخال/الإخراج (I/O) المتزامنة التي تعود إلى القناة نفسها متقارنة وذلك لنمذجة الاتصال بين كائنين. على سبيل المثال، العملية SecureCart في الشكل 5 ـ 3 يمكن أن تكون مرفقة بتسلسل عمليات الإخراج والإدخال userAuthChannel!credential_userAuthChannel?result_^ التي تتطابق مع عمليات الإدخال userAuthChannel?credential_ والإخراج

userAuthChannel?result_^ المنفّذة من قبل مدير المستخدم المخول، وذلك لنمذجة الاتصال بين الكائنين المتطلبين لتخويل المستخدم.

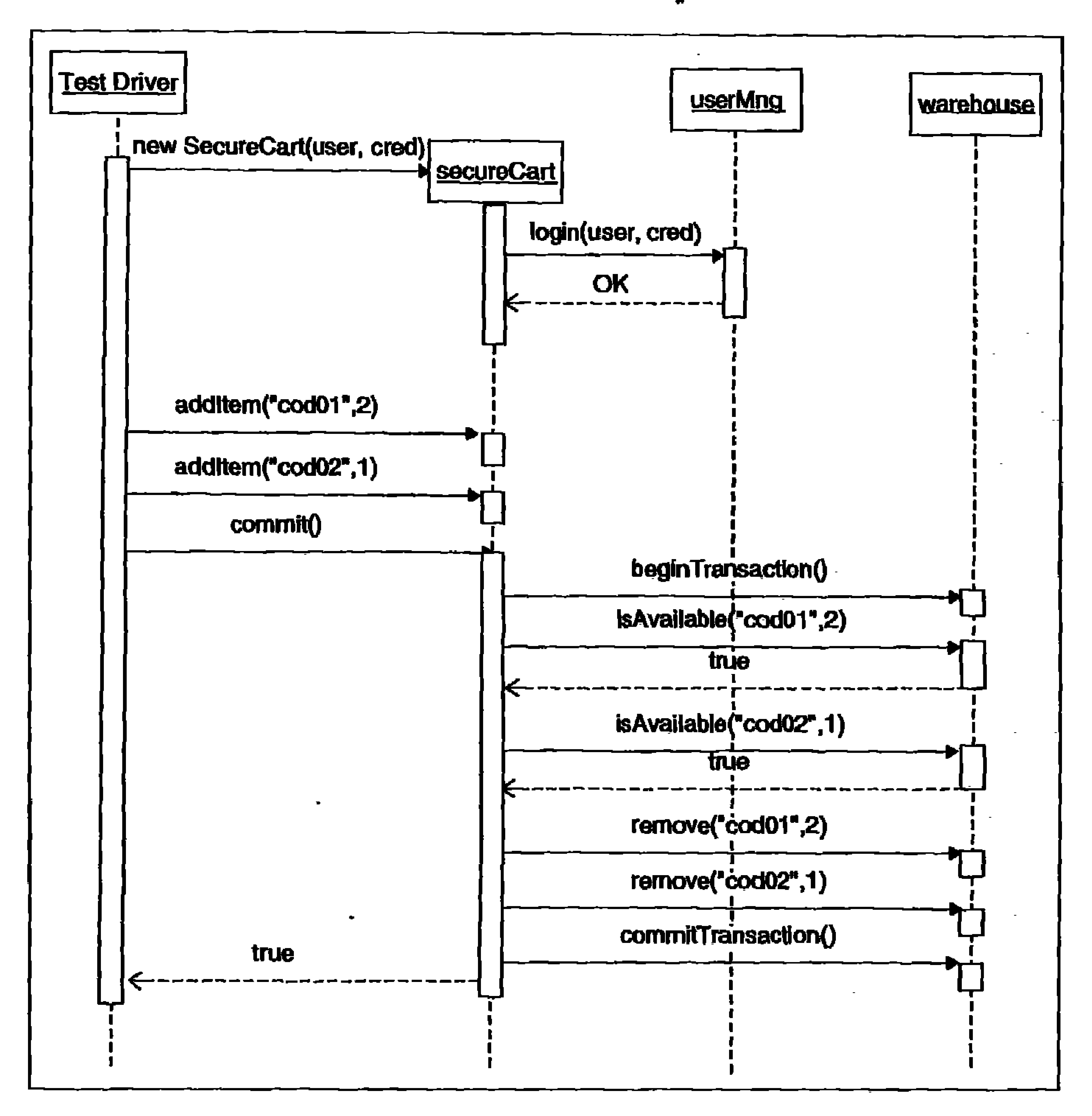
يمكن اشتقاق سيناريوهات الاختبار من مخططات الحالة (Statecharts) للاتصال بواسطة (أ) اختيار مجموعة متكاملة من الكوائن التي يجب أن يتم اختبارها، (ب) تكوين مخططات حالة للاتصال أوتوماتيكياً في مخطط حالة واحدة يمثل سلوك النظام الفرعي المتكامل، و(ج) اشتقاق سيناريوهات الاختبار من مخطط الحالة المتكامل مع التقنيات التقليدية.

حالات مخطط الحالة المكوّن هي عبارة عن المنتج الديكارتي لحالات مخطط الحالة الأصلي. إن حالات الانتقال من حالة إلى أخرى غير المشمولة في الاتصال تربط جميع الحالات التي تعمم الحالات الأصلية المرتبطة بالانتقالات، في حين يتم دمج الانتقالات في عملية انتقال واحدة تكون بديلة عن أزواج الانتقالات المتضمنة في الاتصال. يعرض كلٌ من هارتمان (Hartmann) وإيموبيردورف (Imoberdorf) ومينسينغر (Meisinger)[22] خوارزمية تركيب تزايدية لتكوين مجموعات من مخططات الحالة. تستند الخوارزمية إلى مساعد على الكشف لاختيار مخططات الحالة التي يجب أن تدخل في التركيب وذلك بشكل تزايدي. يهدف الكشف إلى تقليل حجم مخططات الحالة لتصبح متوسطة الحجم وذلك للحد من تأثير تفكك الحالة.

تكمل مخططات التسلسل والتشارك مواصفات مخططات الحالة من خلال وصف التسلسل التفاعلي النموذجي بين الكوائن. يبيّن الشكل 5 ـ 4 مخططاً تسلسلياً يمثل تفاعلاً نموذجياً بين العربة الآمنة ومستخدم بصلاحيات مدير ومستودع ووكيل خارجي. يبيّن المخطط حالة نجاح عملية الدخول إلى النظام متبوعة بإصدار طلب لمادتين.

تشتق مخططات التسلسل عادة من المتطلبات ومواصفات النظام التي يتم تحديدها في مراحل المشروع الأولى، وذلك لنمذجة السيناريوهات التي يمكن تنفيذها والتحقق من صحتها في النظام المستهدف (عادة ما توفّر مخططات التسلسل تفاصيل عن تسلسل عمليات الدمج المتوقعة وقيم البيانات التي يجب تبادلها)، بذا فهي تمثل حالات الاختبار المحتملة. على سبيل المثال، يخصص مخطط التسلسل في الشكل 5 ـ 4 أن العربة يجب أن تستدعي العملية isAvailable مرتين ومن ثم العملية remove مرتين على المستودع. في

الحالتين، يجب أن يمرر الاستدعاء الأول المعاملات «cod01» و«2»، بينما يجب أن يمرر الاستدعاء الثاني المعاملات «cod02» و«1».

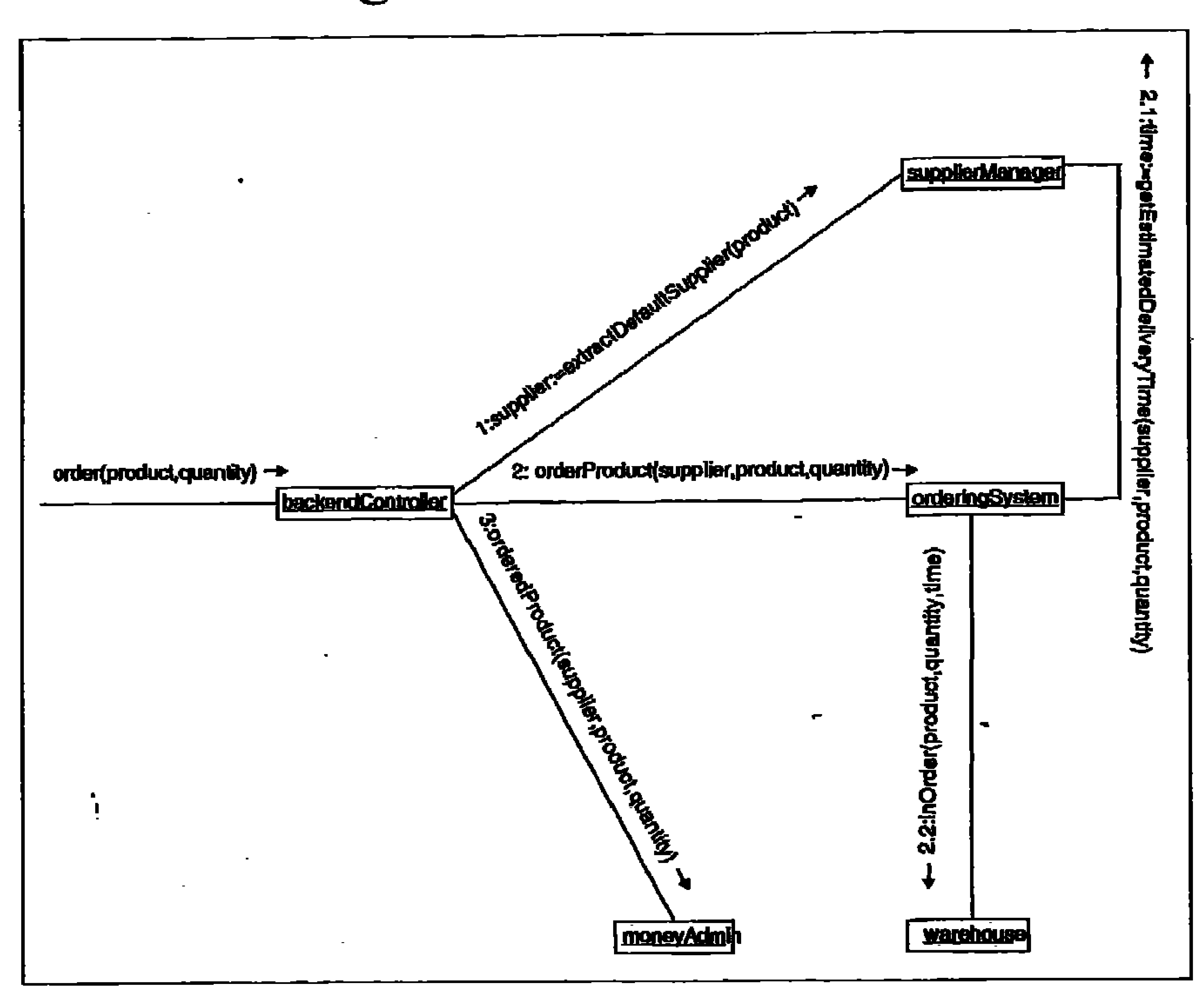


الشكل (5 ـ 4): مخطط تسلسل ينمذج عملية شراء مادتين من العربة الآمنة

تشتق مخططات التسلسل عادة في المراحل الأولى من عملية التطوير، وبذا فهي نادراً ما تكون قابلة للتنفيذ مباشرة، وذلك لأنها تفتقد لتفاصيل التنفيذ. على سبيل المثال، لا يحدد مخطط التسلسل في الشكل 5 ـ 4 أن تنفيذ العملية isAvailable للمستودع تكون «صح» فقط إذا كانت المادة والكمية المطلوبة متوفرتين في قاعدة البيانات. لتنفيذ سيناريو الاختبار المرتبط بهذا المخطط يجب أن يجهز مهندسو الاختبار قاعدة البيانات بطريقة ملائمة.

بشكل عام، لا يمكن إضافة التفاصيل المفقودة أوتوماتيكياً، لكن هناك أدوات تعمل على إنتاج شيفرة تحويل العوامل لتنفيذ مخططات التسلسل ومراقبة سلوك التطبيقات المرتبطة في فترة التنفيذ، وذلك للتحقق ممّا إذا كانت عمليات التنفيذ تلبّي السلوك المحدد في مخططات التسلسل[12]؛ ومن هذه الأدوات SeDiTeC.

أما مخططات التشارك فتوفر طريقة بديلة لوصف تسلسل التفاعلات بين الكوائن. في هذه المخططات، ترتبط الكوائن بخطوط مباشرة تمثل حالات الاتصال. يتم التفاعل بين العمليات المستدعاة. يعنون كل خط بالعملية المرتبطة به أو برسالة مع رقم يشير إلى ترتيب الاتصال. هذا وقد تمثل مخططات التشارك المتغيرات المتبادلة في أثناء الحوسبة. يبيّن الشكل 5 ـ 5 مثالاً على مخطط تشاركي يمثل التفاعلات بين backendController وorderingSystem وsupplierManager وwarehouse وmoneyAdmin لإكمال طلب المواد بنجاح.



الشكل (5 ـ 5): مخطط تشارك يمثل طلب منتج جديد

يعمل backendController على استخلاص المورّد الافتراضي للمنتج

المطلوب، حيث يُطلب المنتج بفعالية عن طريق إصدار طلب لـ orderingSystem الذي يستخلص وقت التسليم المتوقع ويعلم المستودع عن الطلب قيد الانتظار، وأخيراً يقوم backendController بإشعار moneyAdmin لإكمال معالجة الطلب.

تعمل مخططات التشارك ـ كما هو الحال في مخططات التسلسل ـ على نمذجة متسلسلات التفاعل النموذجية، وتشير إلى سيناريوهات الاختبار. يحدد كلٌ من عبد الرازق (Abdurazik) وأوفوت (Offutt)[1] منهجية اختبار بسيطة تتطلب تغطية جميع حالات التشارك المحدد في المخططات.

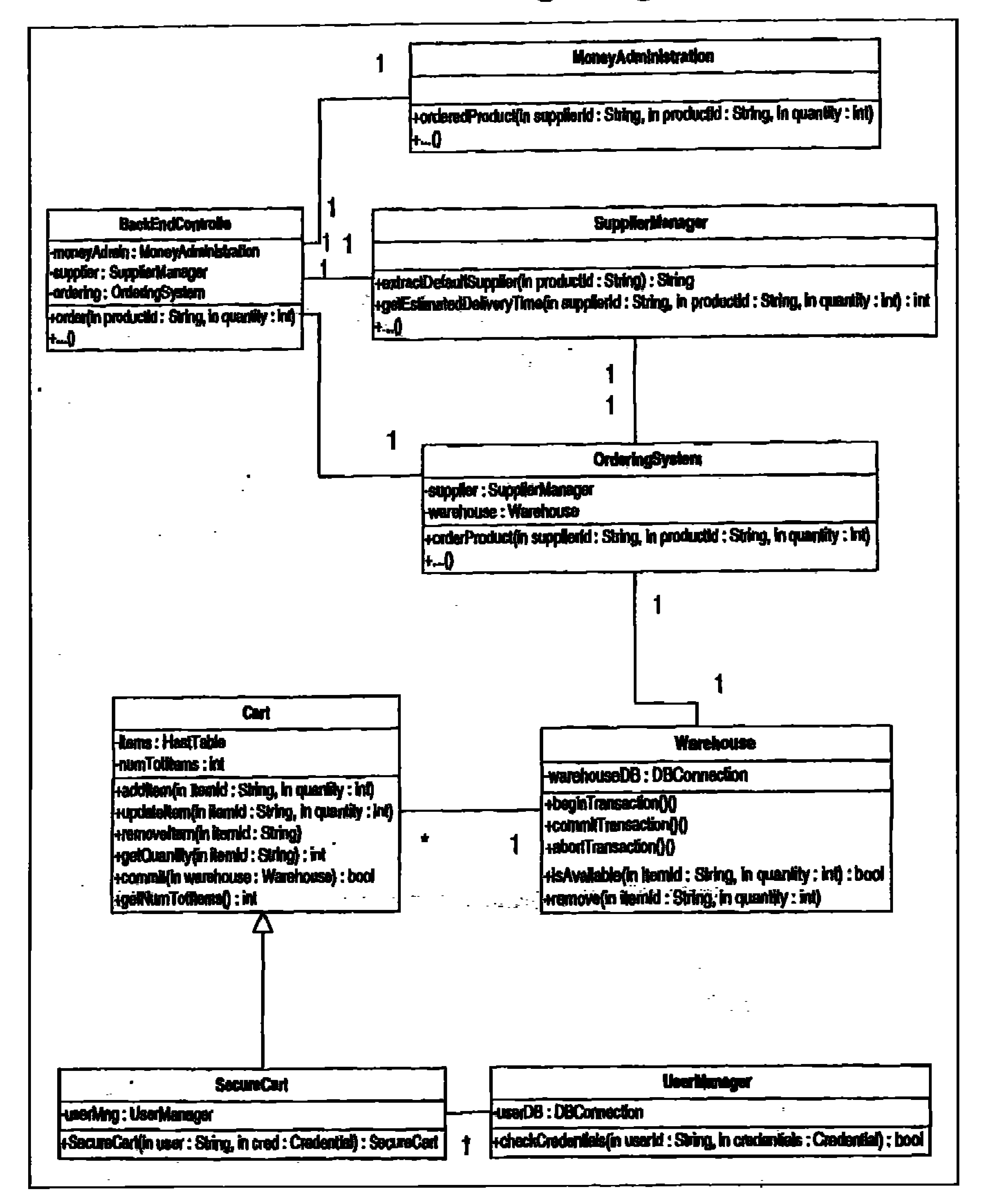
يقترح كلٌ من أندروز (Andrews) وفرانس (France) وغوش (Ghosh) وكريغ (Craig) معايير بديلة بناءً على الظروف التي قد تحدث في المخططات: فهم يتطلبون تغطية جميع الظروف (تغطية ظرفية) وجميع الشروط (تغطية إسنادية كاملة) وكل رسالة على حدة (تغطية كل رسالة في حالة ربط) والمسارات الكاملة التي تمثل مخططات التشارك (تغطية مسارات الرسالة) والرسائل التي تتبادل مجموعات بتغيير حجم المجموعات المتبادلة (تغطية المجموعات)[2].

تحدد مخططات النوع والكائن العلاقات بين الأنواع والكوائن. العلاقات المشتركة الممثلة في مخططات النوع والكائن هي (أ) الاتحادات التي تمثل الروابط بين الأنواع، (ب) التخصيص الذي يمثل الأنواع المتوارثة من أنواع أخرى، و(ج) التراكيب التي تمثل الكوائن التي يحصل عليها من التجميع. يبيّن الشكل 5 ـ 6 مثالاً على مخطط نوع لنظام تسوّق من خلال الإنترنت.

يشتق أندروز والآخرون[2] سيناريوهات الاختبار من مخططات النوع عن طريق تغطية العناصر الهيكلية. فهم يحددون ثلاثة معايير رئيسة: تغطية الارتباط النهائي التعددية وتغطية التعميم وتغطية سمة النوع. تتطلب تغطية الارتباط النهائي التعددية سيناريوهات اختبار تماثل كل ارتباط صفري ووسيط والحد الأقصى من المرات. أما في حالة تعدد الروابط فيتطلب المعيار سيناريوهات اختبار للجداء الديكارتي للسيناريوهات المفردة.

على سبيل المثال، يمكن تغطية الارتباط بين العربة Cart والمستودع Warehouse في الشكل 5 ـ 6 بواسطة ثلاثة سيناريوهات اختبار. يتماثل السيناريو الأول مع مستودع من دون أي ارتباط مع العربة (الحالات الصفرية)،

يتماثل السيناريو الثاني مع المستودع وعربتين (حالات الكميات المتوسطة)، ويتماثل السيناريو الثالث مع مستودع و100 عربة (الحالات القصوى).



الشكل (5 ــ 6): مخطط نوع لنظام تسوّق

يتطلب معيار التعميم سيناريوهات اختبار تماثل كل نمط فرعي من الأنماط مرة واحدة على الأقل. يختبر هذا المعيار إمكانية استبدال نمط كائن بأي من الأنماط الفرعية التابعة له. على سبيل المثال، يمكن تغطية التعميم بين Cart

وSecureCart في الشكل 5 ـ 6 بسيناريوهَي اختبار، بحيث يماثل السيناريو الأول ويستخدم Cart بينما يماثل السيناريو الثاني ويستخدم SecureCart.

يتطلب معيار سمة النوع سيناريوهات اختبار تعمل على جميع مجاميع سمات الحالات لكلّ نوع على حدة. يمكن الحصول على الحالات التي يجب اختبارها بطريقة تقسيم الفئة[31]. على سبيل المثال، يمكن تغطية النوع Cart في الشكل 5 ـ 6 بواسطة إنشاء سيناريوهات اختبار تعمل على تكوين جميع الإعدادات الممكنة لعناصر سمات الحالة وnumTotItems، بناءً على مواصفات العربة.

تكون مواصفات من آلة تحديد الحالة FSM أو Statecharts متوفرة غالباً كما إن هناك أدوات فاعلة جيدة لإنشاء سيناريوهات الاختبار، التي تضمن تغطية كاملة لسلوكيات الكائن بناءً على المواصفات والمعايير المنتقاة. هناك العديد من الخبرات الناجحة التي رواها كل من برايند Briand وكيو Cui ولابايتش Labiche[5] وكل من Hartmann وImoberdorf وMeisinger[22].

هذا وتكون مخططات التسلسل ومخططات التشارك ومخططات النوع متوفرة أيضاً. بذا يمكن إنشاء سيناريوهات الاختبار من هذه المخططات، لكن إكمال اشتقاق مجموعات الفحص يعتمد على دقة هذه المخططات.

لا تتوفر لتقنيات النوع الداخلي فرص كثيرة للعثور على أخطاء في العملية addItem لأنها تهدف إلى كشف أخطاء التكامل، في حين أن أخطاء هذه العملية تكون أخطاء وحدة (unit).

5 ـ 6 تقنيات الاختبار الجبري

تدعم المواصفات الأساسية إنشاء سيناريوهات الاختبار والمشاورات أوتوماتيكياً. اقترح كل من رونك كو دونغ Doong وفرانكل Frankl[11] تقنية جيدة لأتمتة عملية إنشاء سيناريوهات الاختبار من المواصفات الجبرية في نظام ASTOOT. لكن Chen وآخرون قاموا بتوسيع الاقتراح الأصلي ليشمل نظام TACCLE الذي يتعامل مع مجموعة أكبر من السيناريوهات.

يتمّ إنشاء سيناريوهات الاختبار أوتوماتيكياً عن طريق جمع قواعد جبرية بطريقة ملائمة. فإذا أعطي النوع C والتسلسل s لتنفيذ عملية، فإن التسلسل المكافئ يكون عبارة عن تسلسل للتنفيذات التي تنتج منها نفس النتائج بالنسبة إلى الـ s، في حين أن التسلسل غير المكافئ هو الذي ينتج نتائج مختلفة.

السيناريو المكافئ هو زوج من التسلسلات المكافئة، في حين أن السيناريو غير المكافئ هو زوج من التسلسلات غير المكافئة. تبدأ التقنية بمجموعة من سيناريوهات الاختبار للنوع C، ويتم إنشاء سيناريوهات مكافئة وغير مكافئة أوتوماتيكياً من المواصفات الجبرية. يتم تقييم نتائج تنفيذ سيناريوهات الاختبار أوتوماتيكياً بمقارنة نتائج السيناريوهات المكافئة وغير المكافئة.

```
1:  Cart()
2:  addItem(code, qt)
3:  removeItem(code)
4:  updateItem(code, qt)
5:  getQuantity(code)
6:  getNumTotItems()
7:  commit()
```

الشكل (5 ـ 7): واجهة النوع Cart

```
type Cart
syntax
    create: → Cart
    addItem: Cart × String × Integer → Cart
    removeItem: Cart × String → Cart
    updateItem: Cart × String × Integer → Cart
    getQuantity: Cart × String → Integer
    getNumTotItems: Cart → Integer
    commit: Cart → Cart
declare
    C: Cart
    x, y: Integer
    s: String
semantics
     1: getQuantity(create, s) → 0
     2: getQuantity(addItem(C, s, x), s) → x
     3: getQuantity(addItem(C, t, x), s) → getQuantity(C, s)
     4: updateItem(create, s, x) → create
     5: updateItem(addItem(C, s, y), s, x) → addItem(C, s, x)
     6: updateItem(addItem(C, t, y), s, x) → addItem(updateItem(C, s, x), t, y)
     7: removeItem(create, s) → create
     8: removeItem(addItem(C, s, y), s) → removeItem(C, s)
     9: removeItem(addItem(C, t, y), s) → addItem(removeItem(C, s), t, y)
    10: getNumTotItems(create) → 0
    11: getNumTotItems(addItem(C, s, x)) → x+ getNumTotItems(removeItem(C, s))
    12: commit(C) → create
```

الشكل (5 ـ 8): مواصفات النوع Cart

على سبيل المثال، لنفترض النوع Cart الذي يطبّق الواجهة المبيّنة في الشكل 5 ـ 7 والمواصفات الجبرية المبيّنة في الشكل 5 ـ 8. تحدد المواصفات الجبرية جميع التسلسلات الممكنة لتنفيذات العملية. بشكل عام، فإن مجموعة جميع التسلسلات التي يمكن إنشاؤها بتطبيق القواعد الجبرية فقط هي مجموعة لانهائية. يمكننا تقليل مجموعة التسلسلات بأخذ التعبيرات الأساسية فقط ـ أي تلك التعبيرات التي يمكن اشتقاقها مباشرة من البديهيات عن طريق استبدال المتغيرات بالصيغ الطبيعية[8]. لسوء الحظ، يكون عدد التعبيرات الأساسية كبيراً جداً، إذ لا يتاح استخدامها عملياً.

يحد نظام ASTOOT من مجموعة سيناريوهات الاختبار عن طريق تعريف تسلسل تنفيذات العملية مع عمليات[2] غير تحقيقية مختلفة ذات حدوث متكرر، وبمراعاة جميع تراكيب قيم المعامل التي تحدث في المواصفات المشروطة[11]. على سبيل المثال، سيناريوهات الاختبار للنوع Cart هو:

s_{cart} = Cart (), addItem ("001",1), addItem ("002",3), addItem ("003",4), removeItem ("033"(

يُنشئ النظام ASTOOT تسلسلات مكافئة عن طريق تطبيق بديهيات التحويل أوتوماتيكياً على التمثيل الشجري ADT للتسلسل المبدئي (بديهيات التحويل هي جزء من المواصفة الأصلية). على سبيل المثال، بإمكان ASTOOT إنشاء التسلسل المكافئ التالي للنوع الفرعي scart ويحصلّه من النوع scart بواسطة تطبيق البديهيات 7 و8 و9 في الشكل 5-8:

$s_{equivalent}$ = Cart (), addItem ("001",1), addItem ("002",3)

يُنشئ النظام ASTOOT تسلسلات غير مكافئة عن طريق تعديل قيم المعامل للتسلسلات المكافئة لانتهاك الشروط الضرورية. على سبيل المثال، بإمكان ASTOOT إنشاء التسلسل غير المكافئ التالي للنوع الفرعي scart

$s_{non\text{-}equivalent}$ = Cart (), addItem ("001",1), addItem ("002",3), addItem ("003",4) removeItem ("002")

ويتم ذلك بتعديل معامل التنفيذات الأخيرة في $s_{equivalent}$؛ وبناءً على ذلك

(2) العملية التحقيقية للنوع C هي تلك التي تنتج قيمة ولا تعدل من حالة المرحلة التي طبقت عليها.

يتم انتهاك الشرط الذي يتطلب أن يكون مُعامِل العملية removeItem مساوياً لمُعامِل التنفيذات الأخيرة للعملية addItem.

يعمل TACCLE على تنقية التقنية التي يعرضها نظام ASTOOT بواسطة توسيع مفهوم التسلسلات المكافئة[8]. فنظام ASTOOT يراعي أن يكون تسلسلان متكافئين عندما يمكن تحويل أحدهما إلى تسلسل آخر بتطبيق قواعد إعادة كتابة المواصفة. هذا التعريف لا يراعي التسلسلات التي تؤدي إلى حالات مكافئة، حتى لو كانت غير محوّلة إلى تسلسلات أخرى بمراعاة قواعد إعادة الكتابة. يلتقط TACCLE هذه الحالات بمراعاة أن يكون تسلسلان متكافئين إذا كان كلاهما قابلاً للتحويل إلى التكافؤ نفسه[9].

يقارن ASTOOT نتائج تنفيذ السيناريوهات المكافئة وغير المكافئة مع العملية eqn (وقد يكون ذلك متكرراً). يمكن تطبيق العملية eqn بالرجوع إلى مواصفاتها الجبرية أو بمقارنة قيم خصائص الحالة المطبقة. أما العملية eqn المحكومة بالمواصفات فهي أسهل، لكنها قد تحذف بعض تفاصيل التطبيق، بينما تراعي العملية eqn المحكومة بالتطبيق جميع التفاصيل، لكنها قد تفشل عند مقارنة الحالات المتكافئة منطقياً، التي تختلف عن تفاصيل التطبيق غير المرتبطة بالعملية. قد تكون العملية eqn معقدة تماماً بالنسبة إلى الكوائن الكبيرة. أما TACCLE فيقدم سياقات مرتبطة وقابلة للمراقبة لتضييق مجال السلوكيات التي يجب التحقق منها لإثبات تكافؤ الكوائن[8]. حدسياً، يقارن TACCLE حالات الكائن عن طريق مطابقة خصائصه وعن طريق التحقق من نتائج تنفيذات التسلسلات التي تعتمد على الخصائص المختلفة (أي سياق الخصائص المرتبط القابل للمراقبة). تحدد السياقات المرتبطة القابلة للمراقبة عن طريق استخدام مخطط بياني للبيانات ذات الصلة، الذي يمكن بناؤه من الشيفرة المصدرية للنوع[8]. يمكن أن تعمل هذه التقنية على التحقق من التكافؤ بين حالات الكائن أوتوماتيكياً وذلك بتوفير معلومات إضافية من قبل مصممي الاختبار كمجموعة العمليات التحققية.

يمكن أن تكشف التقنيات الجبرية الأخطاء في العربة المبيّنة في الشكل 5-1، إذا كانت استراتيجية الاختبار المبدئية تتضمن حالتي تنفيذ للعملية addItem على الأقل. توفر السيناريوهات المكافئة إجابات تلقائية، التي يمكنها أن تحدد الخطأ إذا تضمّنت حزمة الاختبار سيناريوهات اختبار كاشفة. تقدم التقنيات القائمة على

المواصفات الجبرية أفكاراً مهمة لإنشاء الإجابات الموثوقة التي يمكن استخدامها في السياقات المختلفة.

إن الاختبار المعتمد على المواصفات هو التقنية الأساسية لتصميم سيناريوهات الاختبار. يمكن اشتقاق سيناريوهات الاختبار من النوع «الصندوق الأسود» مبكراً في دورة حياة البرمجية من دون معرفة تفاصيل الشيفرة البرمجية، وتكون فاعلة في الكشف عن العديد من الأخطاء. لكن قد يغفل الاختبار المعتمد على المواصفات بعض أنواع الأخطاء، وتحديداً تلك التي تعتمد على التصميم التفصيلي وخيارات التنفيذ التي لا تنعكس في المواصفات. إضافة إلى ذلك، إن قياس مدى تغطية حزم الاختبار القائمة على المواصفات قد لا يكون سهلاً دائماً، إذ يقترن هذا النوع من الاختبار مع الاختبار المعتمد على الشيفرة البرمجية لالتقاط أخطاء التصميم والتنفيذ ولتوفير مقاييس تغطية بسيطة.

5 ـ 7 الاختبار المحدد بالشيفرة البرمجية

تعّرف معايير الاختبار المحدد بالشيفرة البرمجية من حيث التوصيفات البرمجية التي لم تختبر في أثناء تنفيذ سيناريو الاختبار. تتطلب المعايير الكلاسيكية تغطية البيانات والفروع والشروط والقرارات والمسارات والعلاقات الإجرائية الواجهة[33]، لكنها تهمل السلوك المعتمد على الحالة، وهي بذلك لا تنطبق جيداً على اختبار النوع الداخلي والبيني.

اقترحت معايير هيكلية جديدة للتعامل مع السلوك المعتمد على الحالة. تم تعريف مجموعة من المعايير الهيكلية القائمة على الحالة بواسطة تطبيق معايير اختبار تدفق البيانات الكلاسيكية[32, 21, 14] على بيانات الحالة، وذلك بناءً على تعريفات الحوسبة واستخدامات خصائص الكائن[35, 20, 7]. تراعي معايير اختبار تدفق البيانات التعريفات، واستخدامات خصائص الحالة، أي العبارات التي تؤثر في القيمة المرتبطة بالخصائص أو التي تعتمد على القيمة المرتبطة بالخصائص على التوالي. تقرن معايير تدفق البيانات التعريفات وتستخدم ذلك للإشارة إلى خصائص نفس الكائن (أزواج def-use) وتتطلب تغطية مسارات البرنامج التي تجتاز الأزواج def-use، بمعنى أن المسارات التي تختبر التعريف أولاً، ومن ثم تستخدم نفس الخاصية من دون أن تجتاز تعريفات إضافية للخاصية بين التعريف المُراعى والاستخدام. بهذه الطريقة، فهي تختبر عمليات التنفيذ التي تغير حالة الكوائن قبل غيرها (تعريفات خصائص الكائن) ومن ثم

تستخدم الحالة الجديدة، ما يؤدي إلى فشل محتمل يعتمد على حالات الخطأ. هناك معايير كثيرة لاختبار تدفق البيانات تختلف باختلاف مجموعات أزواج def-use التي تتطلب تغطية جميع أزواج def-use العملية في البرنامج.

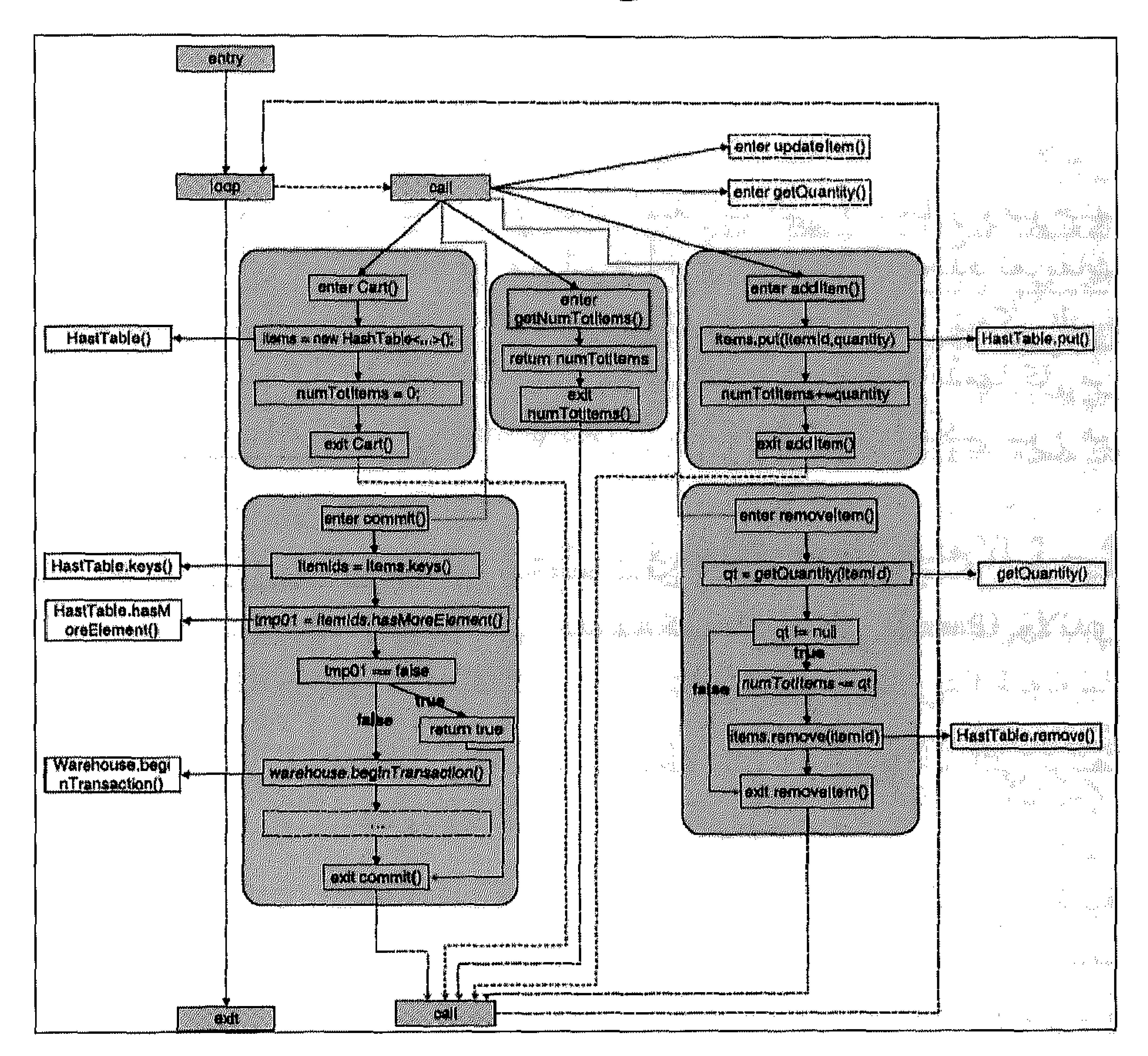
5 ـ 8 الاختبار الهيكلي للنوع الداخلي

تختبر تقنيات النوع الداخلي السلوكيات القائمة على الحالة للأنواع المفردة. فهي تختبر تعريفات واستخدامات المتغيرات التي لا تتجاوز وضوحيتها حدود النوع: أي أنها لا تتفاعل مع أنواع أخرى. يمكننا أن نحدد تعريفات واستخدامات مواصفات الكائن بالرجوع إلى تمثيل مخطط تدفق تحكم النوع CCFG للنوع. يمكن بناء مخطط تدفق تحكم النوع بخوارزمية قدّمها كلٌ من هارولد (Harrold) ورذرميل (Rothermel)[20]. مخطط تدفق تحكم النوع هو رسم بياني يُمثّل نوعٍ مين، حيث:

- تمثّل كل عملية بواسطة مخطط تدفق التحكم، الذي يمكن حوسبته عن طريق الخوارزمية التي قدمها كلٌ من باندي (Pande) ولاندي (Landi) ورايدر (Ryder)[32].
- يمثّل كل استدعاء لعملية بواسطة نقطة استدعاء وإرجاع، التي ترتبط بالعملية المستدعاة ونقاط الخروج على التوالي.
- جميع العمليات العامة تمثل بواسطة متحكم النوع الذي يُنمذج إمكانية استدعاء العمليات العامة بأي ترتيب كان. يتكون متحكم النوع من نقاط إدخال وحلقة واستدعاء وارجاع وخروج. نقاط الاستدعاء والإرجاع ترتبط مباشرة بجميع العمليات العامة؛ أما نقطة الحلقة فترتبط بنقاط الاستدعاء والإرجاع لتمثّل إمكانية تكرار استدعاءات عشوائية عدد معيّن من المرات؛ تمثّل نقاط الإدخال والخروج بداية التنفيذ وانتهاءها.

يبيّن الشكل 5 ـ 9 مخطط تدفق التحكم للنوع Cart الذي أوضحنا شيفرته البرمجية في الشكل 5 ـ 1. تتكون حالة هذا النوع من متغيرين: items وnumTotItems. يمكن استخدام مخطط تدفق التحكم لحساب أزواج def-use بواسطة التأشير أولاً على مخطط تدفق التحكم بالتعريفات والاستخدامات، ومن ثم اجتياز المخطط لحساب الأزواج. على سبيل المثال، بالنسبة إلى المتغير numTotItems التابع للنوع Cart، يمكننا تحديد التعريفات في العمليات Cart

() وcommit () وaddItem () وremoveItem ()، وتحديد الاستخدامات في العملية getNumTotItems () وaddItem () وremoveItem ()، ويمكننا حساب ما مجموعه 12 زوجاً عملياً من أزواج def-use.



الشكل (5 ـ 9): مخطط تدفق تحكم النوع Cart الذي أعطيت شيفرته البرمجية في الشكل 5 ـ 1

يمكن أن تكشف اختبارات تدفق البيانات عن العيوب الموجودة في النوع Cart لأنه أزواج def-use التي يجب شمولها تتضمن تعريف المتغير numTotItems في العملية addItem () واستخدام في العملية addItem (). وكما يحدث دائماً، إن اختبار العبارة الخاطئة في حالة الخطأ لا تضمن أن الخطأ سيكشف، حيث إن حدوث الخطأ قد لا ينتج منه خطأ منظور في جميع حالات التنفيذ.

لنتابع في مثال العربة، المتغير items معرّف في العمليات Cart () وcommit () ويستخدم في العمليات addItem () وremoveItem () وupdateItem () وgetQuantitiy () وcommit () (مرتين في الأسطر 34 و47). ينتج من هذه التعريفات والاستخدامات 12 زوجاً def-use والتي يجب شمولها لتحقيق المعايير. التحليل السابق للمتغير items غير مكتمل ذلك أنها لا تراعي متغيرات العنصر التابع له: المتغير items هو كائن ذو بنية حالة معقدة؛ إذا قمنا بتحليله كمتغير بدائي فإننا نفتقد تأثير استدعاء العمليات الخاصة به. على سبيل المثال، إن استدعاء العملية الموجودة في السطر 10 لا تعتبر كتعريف للمتغير items في التحليل أعلاه حتى لو تغيرت حالة المتغير items.

إن تحليل تدفق البيانات للنوع Hashtable يحسّن من الوضع جزئياً، حيث إن أزواج def-use المحسوبة للنوع Hashtable تنتج متطلبات الاختبار التي تعود إلى عمليات النوع Hashtable[3]، لكنها لا تراعي كيفية استخدام هذه العمليات في النوع Cart.

على سبيل المثال، يمكن تحقيق متطلبات اختبار النوع Hashtable التي تتطلب تنفيذ العملية put متبوعاً بالعملية get بواسطة زوج واحد من عبارات النوع Cart (). على كلٍّ، يتضمن النوع Cart أزواجاً مختلفة عديدة (<10, 29>، <10, 41>، <10, 51>) تستحق أن تختبر، لكن لا يتم اختبارها بما أنها تشير إلى الأزواج نفسها في النوع Hashtable. إن استبدال استدعاءات عمليات الأنواع الخارجية في مخطط تدفق التحكم مع ICFG سينتج منه أزواج def-use إجرائية داخلية، لكنها لم تزل تفتقد التعريفات والاستخدامات الناجمة عن تأثير نوع على آخر.

يمكن أن يتم تحسين تحليل def-use للنوع الداخلي عن طريق احتساب التعريفات والاستخدامات التي تتضمن السياق الذي يعرّف المتغيرات وتستخدم فيه. بهذه الطريقة، على سبيل المثال، سيكون بإمكاننا تحديد جميع أزواج def-use الناجمة عن تأثير النوع Cart على النوع Hashtable.

التقنيات المتأثرة بالسياق موضحة في القسم التالي.

(3) لا تعود متطلبات الاختبار على العمليات، لكنها تعود على تعابير البرنامج. لتبسيط هذا المثال، نحدد متطلبات اختبار النوع Hashtable من حيث العمليات التي تتضمن تعريفات واستخدامات، بدلاً من تعابير برنامج مفرد.

5 ـ 9 الاختبار الهيكلي للنوع البيني

تعنون التقنيات المتأثرة بالسياق اختبار النوع البيني من خلال زيادة التعريفات والاستخدامات بسلاسل من الاستدعاءات التي تقود إلى تعديلات ووصولية للمتغير[35]. يمكن تعريف متغير الحالة o للزوج def-use كزوج من تسلسلات الاستدعاء (d,u) حيث:

$d = CD_1CD_2...CDmD, U = CU_1CU_2...CU_nU, CD_i$

وCUj فهي استدعاءات للعملية. D هي عبارة البرنامج التي تعرف o، أما U فهي عبارة عن برنامج يستخدم المتغير o. هذا مثال على سلسلة استدعاء طولها 3 ناتجة من تنفيذ النوع Hashtable الموضح في الشكل 5 ـ 10، تتكون من Cart-25 وCart-10 وHashtable-6 التي تشير للتعريف المتغير المشتق من تنفيذ السطر السادس في النوع Hashtable الذي ينتج من تنفيذ السطر 10 من النوع Cart الناجم بدوره عن تنفيذ السطر 25 من النوع Cart.

```
1:     public synchronized V put(K key, V value) {
2:          if (value == null) {
3:             throw new NullPointerException();
4:          }
5:
6:          Entry tab[] = table;
7:          int hash = key.hashCode();
```

الشكل (5 ـ 10): مقتطفات من تنفيذ النوع HashTable

إن الزوج def-use (d,u) لمتغير o يتم اختباره عن طريق عملية تنفيذ تتضمن استدعاء التسلسل d متبوعاً بأي تسلسل لاستدعاءات العملية التي لا تعدّل o وتنتهي بتسلسل الاستدعاء u.

يمكن تنفيذ هذا التحليل بمستويات تعمق مختلفة. يعرف ساوتر وبولوك[35] أربعة مستويات تدعى cdu-0 وcdu-1 وcdu-2 وcdu-3. يتطابق المستوى cdu-0 مع تحليل def-use غير مرتبط بالسياق، كما هو مبيّن في القسم 5-8. في هذه الحالة، تتضمن تسلسلات الاستدعاء d وu عبارات البرنامج فقط التي تحدد وتستخدم المتغير. أما المستوى cdu-1 فيشير إلى تسلسلات استدعاء ذات طول 2؛

أي أنها التسلسلات التي تتضمن المستدعي وعبارة البرنامج فقط التي تعدل المتغير وتستخدمه. أما المستويان cdu-2 وcdu-3 فيوسعان التحليل ليشمل الاستدعاءات ذات الأطوال 3 و4 على التوالي.

بالإشارة إلى النوع Hashtable المستخدم من قبل النوع Cart، وكما هو مبيّن في الشكل 5-1، فإن المستوى cdu-1 سيقترن بتعريفات النوع Hashtable التي تشتق من استدعاء putItem عند العبارة رقم 10 من النوع Cart مع استخدامات النوع Hashtable التي تشتق من استدعاءات addItem عند العبارات 29، 34، 41، 47، 51 (تعريفات واستخدامات متغيرات حالة مفردة التي يتم الحصول عليها من تحليل الشيفرة المصدرية للنوع Hashtable).

تقيس معايير اختبار تدفق البيانات اكتمال مجموعات الاختبار لكنها لا توفر عملية لإنشاء سيناريوهات الاختبار. يجمع كلٌ من باي Buy وأورسو Orso وPezzè[7] وPezzè[7] تحاليل تدفق البيانات مع تنفيذ رمزي واختصار أوتوماتيكي لإنشاء سيناريوهات الاختبار التي تحقق معايير تدفق البيانات. تعمل التقنية على احتساب أزواج def-use كما نوقش في القسم 5-7 ومن ثم يتم إنشاء سيناريوهات الاختبار التي تغطي الأزواج المعرفة. يتم إنشاء سيناريوهات الاختبار بخطوتين:

أ) التنفيذ الرمزي الذي يحدد (ا) المسارات التي تغطي الأزواج def-use و(2) شروط المسار المرتبطة (أي قيود متغيرات الحالة التي تسبب تنفيذ المسارات).

ب) الاختصار الأوتوماتيكي الذي ينتج تسلسلات استدعاء كامل بواسطة تكوين شروط المسار لإنشاء سيناريوهات اختبار عملية.

5 ـ 10 الاختبار في ظل وجود التوارث

يتيح التوارث لمهندسي البرمجيات تنفيذ أنواع جديدة كأنواع فرعية تابعة للأنواع المتوفرة أصلاً. الأنواع الفرعية تُعيد استخدام بعض وظائف الأنواع السابقة، وتُعدّل وظائف أخرى، وتُضيف وظائف أخرى، على الرغم من أنه يمكن اختبار الأنواع الفرعية، يمكن كما لو كانت أنواعاً جديدة من حيث المبدأ، إلا أن اختبارات الأنواع القديمة قد تقلل من عدد سيناريوهات الاختبار اللازمة لفحص النوع الفرعي فحصاً تاماً.

على سبيل المثال، النوع SecureCart المحدد بالنوع المبيّن في الشكل 5-6

يوسّع النوع Cart المبيّن في الشكل 5 ـ 10. النوع SecureCart يعيد استخدام جميع عمليات النوع Cart ما عدا constructor الذي أعيد تعريفة ليتطلب هوية المستخدم ومعلومات اعتماده. في هذه الحالة، لا تتأثر العمليات الآتية بالوظائف المضافة، ولا يتطلب الأمر إعادة اختبارها: addItem وremoveItem وgetQuantity وgetNumTotItems وcommit. وبذا يمكننا اختبار النوع SecureCart بواسطة اختبار constructor الجديد فقط.

بشكل عام، يلزمنا إعادة اختبار جميع العمليات التي تتأثر بالعمليات الجديدة أو المعدلة مباشرة أو بشكل غير مباشر. قام كلٌ من Harrold وماكغريغور McGregor وفيتسباتريك Fitzpatrick[19] بجمع سيناريوهات الاختبار الهيكلية والوظيفية حسب العمليات التي يتم اختبارها ونوع الاختبار (وحدة أو تكامل)، وقاموا بتحديد سيناريوهات الاختبار التي يجب إضافتها إلى حزمة الاختبار وإلى السيناريوهات التي يجب أن يعاد تنفيذها حسب العمليات المضافة والمعدّلة في النوع الفرعي.

تتطلب العمليات الجديدة المضافة في النوع الفرعي سيناريوهات اختبار وحدة جديدة وتكاملاً (إذا كانت تتفاعل مع توصيفات أخرى). تتطلب العمليات التلخيصية سيناريوهات اختبار وظيفية فقط، بينما تتطلب العمليات الأساسية سيناريوهات اختبار وظيفية وهيكلية.

العمليات التي لم تتغير (أي العمليات المتوارثة من النوع الأم من دون تغيير) لا تتطلب سيناريوهات اختبار إضافية. يجب أن يعاد تنفيذ سيناريوهات اختبار التكامل إذا كان النوع الفرعي يعدل التكاملات بعملية معادة الاستخدام.

أما العمليات المعاد تعريفها، أي تلك التي غيرها النوع الفرعي، فتتطلب إضافة سيناريوهات اختبار وظيفية وهيكلية جديدة إلى السيناريوهات المتوفرة التي يتوجب إعادة تنفيذها. إذا كانت العملية تتفاعل مع توصيفات أخرى، علينا أن نشتق سيناريوهات اختبار التكامل وسيناريوهات اختبار الوحدة.

5 ـ 11 اختبار الانحدار

في أثناء عملية التطوير، يُنتج مهندسو البرمجيات إصدارات عديدة من الأنواع التي تكوّن النظم البرمجية. كما في حالة التوارث، يتم اختبار جميع الإصدارات كما لو كانت أنواعاً جديدة من دون مراعاة سيناريوهات الاختبار

المشتقة للإصدارات السابقة، وهذا يسبّب هدراً في الوقت والموارد. تستخدم تقنيات اختبار الانحدار معلومات عن التغيرات التي تتم في الإصدارات الجديدة، وذلك لتحديد سيناريوهات الاختبار التي يجب أن يعاد تنفيذها لضمان أن الإصدارات الجديدة لا تتضمن عيوباً. تركز تقنيات اختبار الانحدار على الوظائف التي يجب أن لا تتأثر بالتغيرات والوظائف المُضافة أو المعدلة في الاصدارات الجديدة التي قد تتطلب سيناريوهات اختبار جديدة يمكن إضافتها إلى حزمة الانحدار، ويمكن إنتاجها حسب التقنيات الموصوفة مسبقاً في هذا الفصل.

إن منهجيات اختبار الانحدار البسيطة ستتطلب إعادة تنفيذ جميع سيناريوهات الاختبار المشتقة لجميع الإصدارات السابقة، وبذا تقلّل من جهد تصميم الاختبار، لكن لا تقلل من زمن تنفيذ الاختبار. يمكن أن تؤدي هذه المنهجيات إلى تنفيذ العديد من سيناريوهات الاختبار حتى بالنسبة إلى التغيرات والنطاق المحدودة. اقترح العديد من الباحثين تقنيات لاختيار مجموعات جزئية من سيناريوهات اختبار الانحدار للبرمجيات كائنية التوجه(18، 24، 25، 34). بعض هذه التقنيات آمنة، بينما بعضها الآخر غير آمن. تضمن تقنيات اختبار الانحدار الآمنة أن المجموعة الجزئية المُنتقاة من سيناريوهات الاختبار لن تغفل أيّاً من العيوب التي يمكن أن تكشفها مجموعة سيناريوهات الاختبار الأصلية، بينما قد لا تكشفها التقنيات غير الآمنة.

المنهجيات الشائعة المستخدمة في النظم كائنية التوجه تستند إلى تعريف الجدار الناري firewall الذي يمثّل مجموعة من الأنواع التي يمكن أن تتأثر بالتغيرات التي تطرأ على الإصدارات الجديدة(24، 25، 36). يتم حوسبة جدار الحماية بناءً على التغيرات التي يمكن أن تحدد أوتوماتيكياً بواسطة مقارنة الشيفرة المصدرية لإصدارين مختلفين، بينما يمكن حوسبة مجموعة الأنواع التي يمكن أن تتأثر بالتغيرات بمراعاة التوارث والانحدار والعلاقات المرتبطة (بعض المنهجيات تأخذ في الاعتبار الروابط متعددة الأشكال والروابط الديناميكية). تقارب معظم العمليات جدار الحماية عن طريق مراعاة بعض العلاقات فحسب. التحديد الجيد لجدار الحماية هو ما يضمن الأمن فقط، لكن التحديد الجيد غالباً ما يكون مكلفاً. إن مجموعة سيناريوهات اختبار الانحدار التي يجب إعادة تنفيذها هي مجموعة السيناريوهات التي تنفّذ الأنواع التي تنتمي إلى جدار الحماية لجميع الأنواع المعدلة. إن جدار الحماية المحوسب وتحديد

سيناريوهات الاختبار التي يجب إعادة تنفيذها بسيطة. على كلٍّ، لا تكون التقنية المستخدمة فاعلة دائماً: فبعض التغيرات يمكن أن تختار جميع سيناريوهات الاختبار من الحزمة الأصلية، وبذا يقل الزمن اللازم للتنفيذ. وبناءً على ذلك، يجب أن تستخدم هذه التقنية للتغيرات الصغيرة والمتزايدة.

ثمة منهجيات أخرى تختار حزم الانحدار للبرمجيات كائنية التوجه من معلومات تدفق التحكم. يقترح كلٌّ من Rothermel وHarrold وديدهيا Dedhia[34] تقنيات ترتكز على مخطط تدفق تحكم النوع CCFG. أما الفروقات بين الأنواع، فهي محددة عن طريق مقارنة مخططات تدفق تحكم النوع الخاص بها. تكون مخططات تدفق تحكم النوع مرفقة بحواشي عن معلومات التغطية - أي أنها تكون مرفقة بمجموعة من سيناريوهات الاختبار التي تغطي الحواف موفِّرة بذلك المعلومات اللازمة لتختار أوتوماتيكياً السيناريوهات التي يجب أن يعاد تنفيذها. هذه التقنية آمنة، كما أنها تعمل في حال اختبار النوع الداخلي، لكنه لا يجدول اختبار النوع الداخلي فورياً، كما أن هناك مزايا كالروابط الديناميكية قد تعقد عملية التحليل بشدة.

تم التحقق من الفكرة الأساسية التي يقدمها المرجع[34] من قبل Harrold وآخرين[18]، حيث قاموا بعنونة برامج جافا التي تمثلها مخططات النوع الداخلي لجافا JIG. مخططات النوع الداخلي لجافا تمثل جميع مزايا لغة البرمجة جافا بما في ذلك معالجة الاستثناءات والتطبيقات غير المكتملة، وبذا فهي تدعم تحليل العديد من المزايا التي لا تتوفر في مخطط تدفق تحكم النوع.

5 ـ 12 الاستنتاجات

التصميم كائني التوجه يُقلّل من تكرار حدوث بعض أنواع الأخطاء التقليدية، لكنه يكشف مشكلات جديدة لم يتم كشفها بشكل كافٍ بواسطة الاختبارات وتقنيات التحليل التقليدية. تساعد الحلول العديدة على فهم منظور المشكلات وتحدد بعض المنهجيات المفيدة. إلى غاية الآن، ركزت الأبحاث على بعض المشكلات، على سبيل المثال، تلك التي تشتق من السلوك القائم على الحالة والتضمين والتوارث وتعدد الأشكال وعدم كشف العديد من المشكلات الكبرى الأخرى، مثل المشكلات التي تشتق من استخدام العموميات والاستثناءات.

لكن ما هو أكثر أهمية، الحلول التي لا توفر إطار عمل كامل وناضج لاختبار النظم كائنية التوجه.

يعمل الباحثون بنشاط على التحقق من المنهجيات الجديدة للتغلب على مجموعة المزايا كائنية التوجه بأكملها، إضافة إلى الموازنة جيداً بين اختبار النظام الوحدوي والتكاملي. يعمل المتمرسون في هذا المجال على جمع البيانات عن مدى حدوث العيوب وتوزيعها في البرمجيات كائنية التوجه؛ وهم يتفهمون التحديات الجديدة لاختبار النظم كائنية التوجه أكثر من ذي قبل، وهم يدينون بفضلٍ للحلول التي تقدمها مجموعات البحث.

إن هذا الإدراك المُتنامي للمشكلات والحلول ستؤدي قريباً إلى ظهور جيل جديد من أدوات CASE التي ستتعامل مع اختبار وتحليل النظم كائنية التوجه بكفاءة.

المراجع

1. A. Abdurazik and J. Offutt. «Using UML collaboration diagrams for static checking and test generation.» In A. Evans, S. Kent, and B. Selic (eds.). paper presented at: *Proceedings of the Third International Conference on the Unified Modeling Language*, LNCS 1939. Berlin: Springer, 2000, pp. 383-395.

2. A. Andrews [et al.]. «Test adequacy criteria for UML design models.» *Software Testing, Verification and Reliability*: vol. 13, Hoboken, NJ: John Wiley & Sons, 2003, pp. 95- 127.

3. R. V. Binder. *Object-Oriented Systems: Models, Patterns and Tools.*, Reading, MA: Addison-Wesley, 1999.

4. G. V. Bochmann and A. Petrenko. «Protocol testing: Review of methods and relevance for software testing.» paper presented at: *Proceedings of the 1994 ACM SIGSOFT International Symposium on* Software Testing and Analysis. New York: ACM Press, 1994, pp. 109-124.

5. L. C. Briand, J. Cui, and Y. Labiche. «Towards automated support for deriving test data from UML statecharts.» In P. Stevens, J. Whittle, and G. Booch editors. paper presented at: *Proceedings of the* International Conference on the Unified Modeling Languages and Applications, LNCS 2863, Berlin: Springer, 2003, pp. 249-264.

6. L. C. Briand, Y. Labiche, and Y.Wang. «Using simulation to empirically investigate test coverage criteria based on statechart.» paper presented at: *Proceedings of the 26th International Conference on Software Engineering.* New York: IEEE Computer Society, 2004, pp. 86-95.

7. U. Buy, A. Orso, and M. Pezzè. «Automated testing of classes.» paper presented at: *Proceedings of the 2000* ACM SIGSOFT International Symposium on Software Testing and Analysis. New York: ACM Press, 2000, pp. 39-48.

8. H. Y. Chen [et al.]. «In black and white: An integrated approach to class-level testing of object-oriented programs.» *ACM Transactions on Software Engineering* and Methodology (TOSEM): vol. 7, no. 3, 1998, pp. 250-295.

9. H. Y. Chen, T. H. Tse, and T. Y. Chen. «TACCLE: A methodology for object-oriented software testing at the class and cluster levels.» *ACM Transactions on Software Engineering and* Methodology (TOSEM): vol. 10, no. 1, 2001, pp.56-109.

10. T. S. Chow. Testing design modeled by finite-state machines. *IEEE Transactions on Software* Engineering: vol. 4, no. 3, 1978, pp. 178-187.

11. R.-K. Doong and P. G. Frankl. «The ASTOOT approach to testing object-oriented programs.» ACM Transactions on Software Engineering and Methodology (TOSEM): vol. 3, no. 2, 1994, pp. 101-130.

12. F. Fraikin and T. Leonhardt. SeDiTeC: testing based on sequence diagrams. paper presented at: *Proceedings* of the International Conference on Automated Software Engineering (ASE). IEEE Computer Society, New York, 2002, pp. 261-266.

13. P. G. Frankl and E. J. Weyuker. «An Applicable Family of Data Flow Testing Criteria.» *IEEE Transactions on Software Engineering*: vol. 14, no. 10, 1988, pp. 1483-1498.

14. P. G. Frankl and E. J.Weyuker. «An analytical comparison of the fault-detecting ability of data flow testing techniques.» paper presented at: *Proceedings of the 15th International Conference on Software Engineering.* New York: IEEE Computer Society Press, 1993, pp. 415-424.

15. S. Fujiwara [et al.]. «Test selection based on finite state models.» *IEEE Transactions on Software Engineering*: vol. 17, no. 6, 1991, pp. 591-603.

16. J. A. Goguen and J. Meseguer. «Unifying functional, object-oriented, and relational programming with logical semantics.» In: *Research Directions in Object-Oriented Programming*. Cambridge, MA: MIT Press, 1987, pp. 417-477.

17. D. Harel. Statecharts: «A visual formalism for complex systems.» *Science of Computer Programming*: vol. 8, no. 3, 1987, pp. 231-274.

18. M. J. Harrold [et al.]. «Regression test selection for java software.» paper presented at: *Proceedings of the ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications*. New York: ACM Press, 2001.

19. M. J. Harrold, J. D. McGregor, and K. J. Fitzpatrick. «Incremental testing of object-oriented class structures.» paper presented: *Proceedings of the 14th International Conference on Software Engineering*. New York: IEEE Computer Society Press, 1992, pp. 68-80.

20. M. J. Harrold and G. Rothermel. «Performing data flow testing on classes.» paper presented at: *Proceedings of the 2nd ACM SIGSOFT Symposium on Foundations of Software Engineering*. New York: ACM Press, 1994, pp. 154-163.

21. M. J. Harrold and M. L. Soffa. «Efficient computation of interprocedural definition-use chains.» *ACM Transactions on Programming Languages and Systems*: vol. 16, no.2, 1994, pp.175-204.

22. J. Hartmann, C. Imoberdorf, and M. Meisinger. «UML-based integration testing.» paper presented at: *Proceedings of the 2000 International Symposium on Software Testing and Analysis (ISSTA)*. New York: ACM Press, 2000, pp. 60-70.

23. C. A. R. Hoare. *Communicating Sequential Processes*. Englewood Cliffs, NJ: Prentice-Hall, 1987.

24. P. Hsia [et al.]. «A technique for the selective revalidation of OO software.» *Journal of Software Maintenance: Research and Practice*: vol. 9, no. 4, 1997, pp.217-233.

25. D. C. Kung [et al.]. «Change impact identification in object oriented software maintenance.» paper presented at: *Proceedings of the International Conference on Software Maintenance*. Washington, DC: IEEE Computer Society, 1994, pp. 202-211.

26. D. Lee and M. Yannakakis. «Principles and methods of testing finite state machines: A survey.» paper presented at: *Proceedings of the IEEE*: vol. 84, no. 8, 1996, pp. 1090-1123.

27. B. Marick. *The Craft of Software Testing: Subsystems Testing Including Object-Based and Object-Oriented Testing*. Englewood Cliffs, NJ: Prentice-Hall, 1997.

28. B. Meyer. *Object-Oriented Software Construction*, 2nd ed. Prentice-Hall International Series in Computer Science. Englewood Cliffs, NJ: Prentice-Hall, 2000.

29. Object Management Group (OMG). Unified modeling language: Infrastructure, v2.0. Technical Report formal/05-07-05, OMG, March 2006.

30. A. J. Offutt, Y. Xiong, and S. Liu. Criteria for generating specification-based tests. paper presented at: *Proceedings of the Internationcal Conference on Engineering of Complex Computer Systems (ICECCS)*, Washington, DC: IEEE Computer Society, 1999, pp. 119-129.

31. T. J. Ostrand and M. J. Balcer. «The category-partition method for specifying and generating functional tests.» *Communications of the ACM:* vol. 31, no. 6, 1988, pp. 676-686.

32. H. D. Pande, W. A. Landi, and B. G. Ryder. «Interprocedural def-use associations for C systems with single level pointers.» *IEEE Transactions on Software Engineering*: vol. 20, no. 5, 1994, pp. 385-403.

33. M. Pezzè and M. Young. *Software Test and Analysis: Process, Principles and Techniques*. Hoboken, NJ: John Wiley & Sons, 2006.

34. G. Rothermel, M. J. Harrold, and J. Dedhia. «Regression test selection for C+ + software.» *Journal of Software Testing, Verification and Reliability*: vol. 10, no. 6, 2000, pp. 77-109.

35. A. L. Souter and L. L. Pollock. «The construction of contextual def-use associations for objectoriented systems.» *IEEE Transactions on Software Engineering*: vol. 29, no. 11, 2003, pp. 1005-1018.

36. L. White and K. Abdullah. «A firewall approach for the regression testing of object-oriented software.» In: *Proceedings of the Software Quality Week*, 1997.

6

لغة النمذجة الموحدة والأساليب النظامية: دراسة حالة استخدام

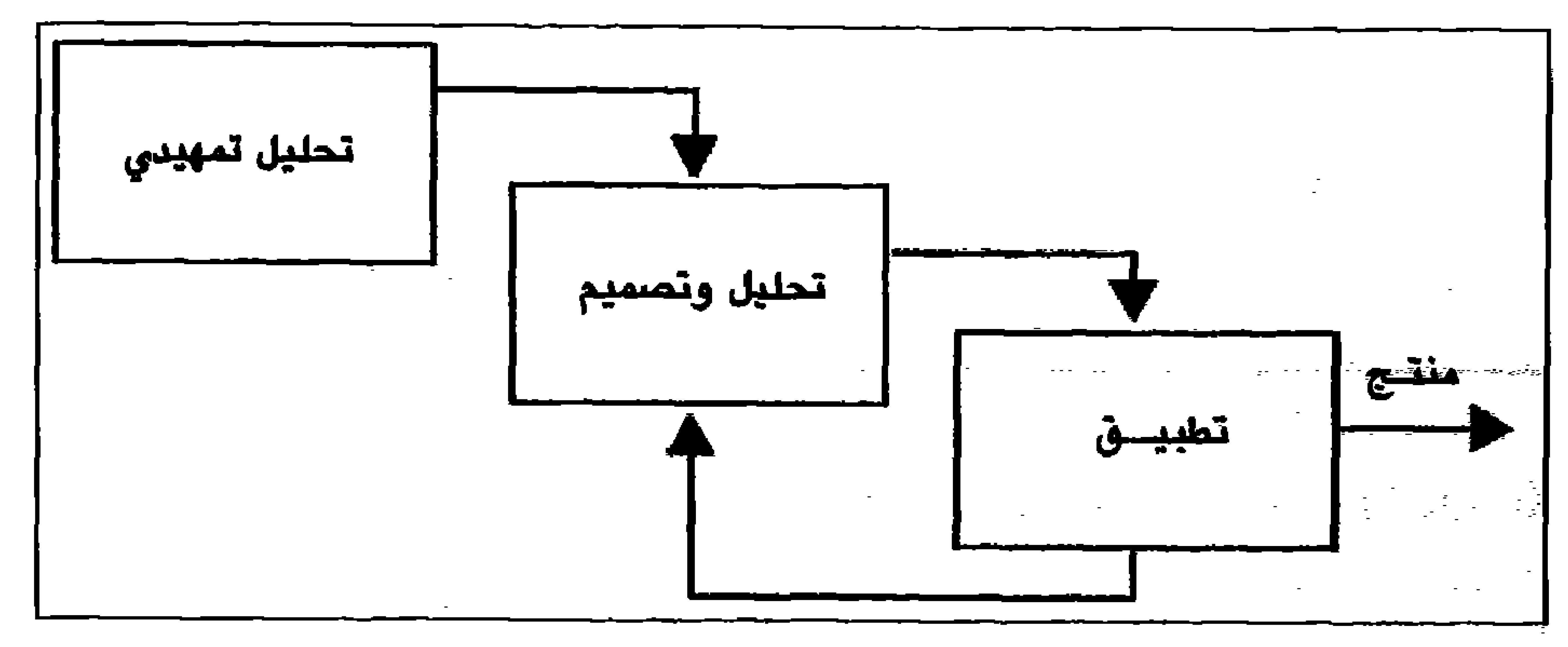
كارلو مونتانغيرو (Carlo Montangero)

6 ـ 1 مقدمة عامة

معظم المنهجيات المعروفة لتطوير البرمجيات تطورية في الوقت الراهن، بما في ذلك الأساليب السريعة Agile[2] والعمليات الموحدة (UP)؛ أي إنها عبارة عن منهجيات دورية، كما هو موضح في الشكل 6-1. بعد التحليل التمهيدي الذي يستهدف الجدوى الإجمالية وتقييم المخاطر، تدخل عملية التطوير دورة مقسمة إلى فترات برمجية دورية iterations يتم في كلٍّ منها تحليل وتصميم وبناء جزء من النظام (كخاصية معيّنة أو سيناريو استخدام، إلى غير ذلك)، ما يتيح للمُستخدم إمكانية استخدام أجزاء متتابعة من النظام يزيد عدد الوظائف والمزايا والجودة في كل منها على سابقه. تكمن الميزة الأساسية لهذه المنهجية في المرونة العظيمة في التجاوب مع التغيير الطارئ على المتطلبات الذي تفرضه الطبيعة المتطورة لبيئات الأعمال الحالية.

تختلف المرحلتان اللتان تتكون منهما الدورة في أن التنفيذ يركّز على الشيفرة البرمجية، ويكون نطاق العمل صغير (أي المتطلبات المطلوب تنفيذها في الدورة الواحدة)، نظراً إلى أن الشيفرة البرمجية تكتب حسب بعض المواصفات، بينما يركّز التحليل والتصميم على عدد كبير ومتنوع من النماذج ويكون النطاق في هذه الحالة أكبر، إذ يتم إنتاج النماذج بناء على المتطلبات ككل. النماذج التي يتم بناؤها في مرحلتي التحليل والتصميم تكون عبارة عن

مستوعبات للمعرفة التي يتم تحصّيلها عن النظام والغرض المرجو من بنائه. تتيح هذه النماذج للمطوّرين الوصول إلى فهم أعمق عن مزايا النظام قبل وقت كافٍ من بنائه. كما أنها أساس عملية التصميم المشتركة التي تعتبر جوهر تطوير البرمجيات الحديثة.



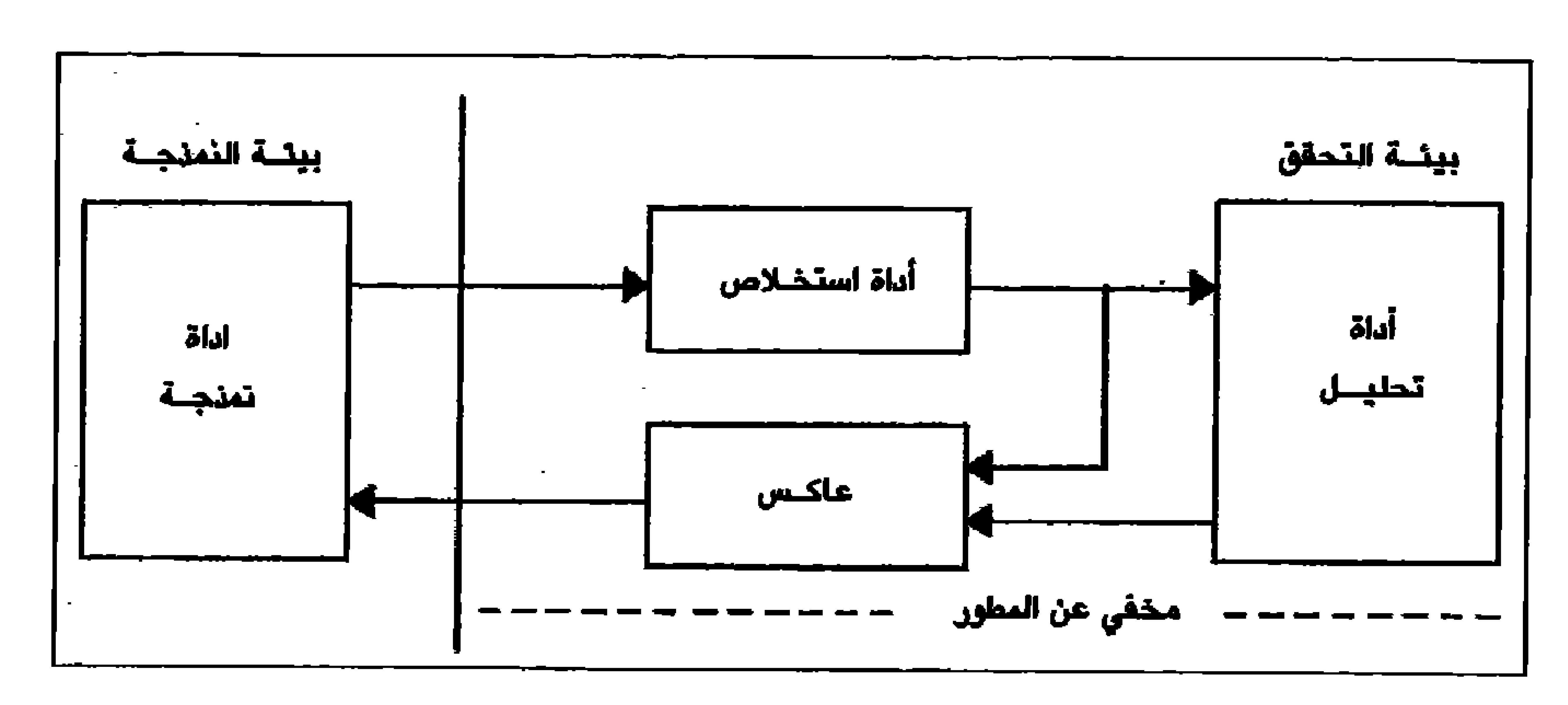
الشكل (6 ـ 1) : دورة حياة التطور

المرحلتان متشابهتان أيضاً حيث إنهما ترتكزان على حلقات بناء ومراجعة وإصلاح تكرارية أضيق. لذا، فهما تختلفان في المعطيات التي تعملان عليها لكنهما متشابهتان في العملية (عمل وإصلاح)؛ هذا وتعتمد كفاءة عملية التطوير على درجة الدعم الأوتوماتيكي بدرجة كبيرة. في هذه الأيام، يتم دعم المرحلتين بدرجات مختلفة في هذا الصدد. تكون عملية التنفيذ تامة طالما أنها مرت بفترة بحث وبرمجة طويلة، ويتم دعمها من قبل أدوات وفيرة كأدوات التجميع وأدوات التدقيق وأدوات التنقيح والبناء وأدوات الربط للاختبار. كما إن هناك عدداً من الأدوات المستخدمة لدعم التحليل والتصميم تم تطويرها من قبل لجنة البحث عن الأساليب الرسمية، كما يتم تطوير المزيد من الأدوات. لكن تبنّي هذه الأدوات من قبل الممارسين ليس متقبلاً بصورة واسعة. يرتبط جزء من المشكلة بالتنوع العظيم للمنهجيات، حيث لكلٍ منها الشكليات الخاصة بها وأهداف يرجى تحقيقها.

إضافة إلى ذلك، قد يكون مستوى التشكيل نفسه مشكلة، ذلك أنه قد يستلزم منحنى تعليم شديد الانحدار حيث يتطلب درجة تعليم لا تكون متوفرة في الأغلب في بيئات البرمجة المتوسطة.

اتخذ مشروع DEGAS[7] خطوات لتعزيز الاستخدام المباشر لأدوات

التحليل من قبل مصممين ذوي مستوى متوسط. تكمن الفكرة الموضحة في الشكل 6-2 في إتاحة المجال أمام المطوّرين أن يستخدموا البيئة التطويرية الخاصة بهم في أثناء إجراء عمليات التحليل في بيئة التحقق الخاصة بها. فحتى يتم تحليل نماذج التطوير في بيئة التحقق، يجب البدء باستخدام أداة استخلاص (Extractor) تعمل على استخلاص أجزاء من النموذج الذي سيكون مرتبطاً بالتحليل ويضعها في بيئة التحقق. بعد أن تكتمل عملية التحليل، تكون نتائج التحليل متوفرة للمطوّر باستخدام عاكس (Reflector). غالباً ما يتم عرض النتائج كتنسيق مرتب للنموذج الأصلي. حتى تكون هذه المنهجية عملية من وجهة نظر المطوّر، يجب أن تكون أداة الاستخلاص وأدوات التحليل والعاكس مؤتمتة ومخفية عن المطوّر. وهكذا، لن يحتاج المطوّر إلى معرفة التفاصيل الأدق للعناصر، لكن قد يركّز على تصميم النظام.



الشكل (6 ـ 2) : مشروع DEGAS

في مشروع DEGAS، تستثمر بيئة التطوير لغة البرمجة الموحدة UML[16]، UML[16]، إذ أصبح لها مؤخراً تأثير في العديد من التطبيقات المستخدمة في الحياة اليومية نظراً إلى شعبيتها المتزايدة. من بين الميزات التي يحققها استخدام لغة النمذجة الموحدة أن هناك توكيداً على العروض الرسومية مع القدرة على تعديلها باستخدام أدوات الطباعة ومنهجية العرض ودعم النقض والتناقض (مؤقتاً). تشجع هذه الخصائص الحوار مع المعنيين في المشروع حتى لو كانوا ذوي معرفة تقنية قليلة. في الوقت نفسه، تكون اللغة دقيقة بما فيه الكفاية لتتيح للمصمم التعبير عن كافة المعلومات الضرورية لاستخلاص النماذج الأساسية اللازمة للتحليل. من ميزات لغة النمذجة الموحدة الجذابة الأخرى هي الخيارات

الكبيرة لبيئات الأعمال التجارية والدعم الحر، إضافة إلى أن لغة النمذجة الموحدة تندمج بسهولة في حزمة واحدة من خيارات منهجيات النمذجة المؤسسة جيداً المتعارف عليها بين المشاركين.

تستخدم بيئة التحقق في مشروع DEGAS حسابات العمليات، وهي عبارة عن نماذج السلوك الخاصة بالنظم، وسيركّز تحليل هذه الحسابات على المظاهر السلوكية للنظم، وبذا تتم عملية تحليل مظاهر بنية النظام بحيث تطبع هيكليات الكائن بشكل جيد وتكون المخططات متسقة داخلياً، وهذه هي أنواع التحليل التي تنفّذ في لغة النمذجة الموحدة في الوقت الحاضر. أما النتيجة الرئيسة العملية لمشروع DEGAS فهي الأداة Choreographer وهو منصة تصميم تكاملية لنمذجة النظم البرمجية النوعية والكمية[4]. يتم نشر التحليل النوعي للتحقق من أمن بروتوكولات الاتصالات المستخدمة في التطبيق. يضمن هذا التحليل عدم وجود محاولات اختراق ناجحة لخاصية المصادقة على الرسائل المتبادلة شريطة أن لا يكون هناك أي اختراقات ناجحة للنظام الأساسي المرمّز المستخدم لحماسة الرسائل. في حال تمّ اختراق عملية المصادقة، تشير عملية التحليل إلى النطاقات التي أختُرقَت. التحليل الكمي الذي ينفذ هو عبارة عن تحليل لأداء نموذج النظام. وهذا من شأنه أن يحدد المكوّنات غير المستغلة أو تلك المستغلة بإفراط، الذي يدل على وجود توزيع ضعيف للموارد الحاسوبية. يُعنى ما تبقى من هذا الفصل بتحليل الأمن والحماية. نتمسك بوجهة نظر مطوّر النظم، ولم نعد نخوض في الأساس النظامي للعمليات الرياضية للمنهجية. بإمكان القارئ المهتم مطالعة المرجع[5] لقراءة المقدمة العامة والمرجع[3] لقراءة مناقشة مستفيضة. هذا وقد تم استعراض التحليل الكمي في المرجع[4]، ونوقش في المرجع[10].

يلقي القسم التالي الضوء على بعض مظاهر لغة البرمجة الموحدة ذات الصلة بنقاشنا؛ بإمكان القارئ ذي المعرفة الجيدة بالموضوع تخطّي هذا القسم. ثم سنعرض ForLySa وهو إطار عملي، في Chreographer، يدعم المصمم في نمذجة البروتوكولات التي يجب تحليلها للوقوف على خروقات المصادقة. نناقش كيفية استغلال إطار العمل لبناء نموذج ديناميكي للبروتوكول في سياق النموذج الثابت ما يوفر جميع المعلومات اللازمة للتحليل. قبل عرض الاستنتاجات، نناقش كيفية عرض نتائج التحليل على المصمّم، وكيف يقوم بتفسيرها في حال وجود خلل في البروتوكول.

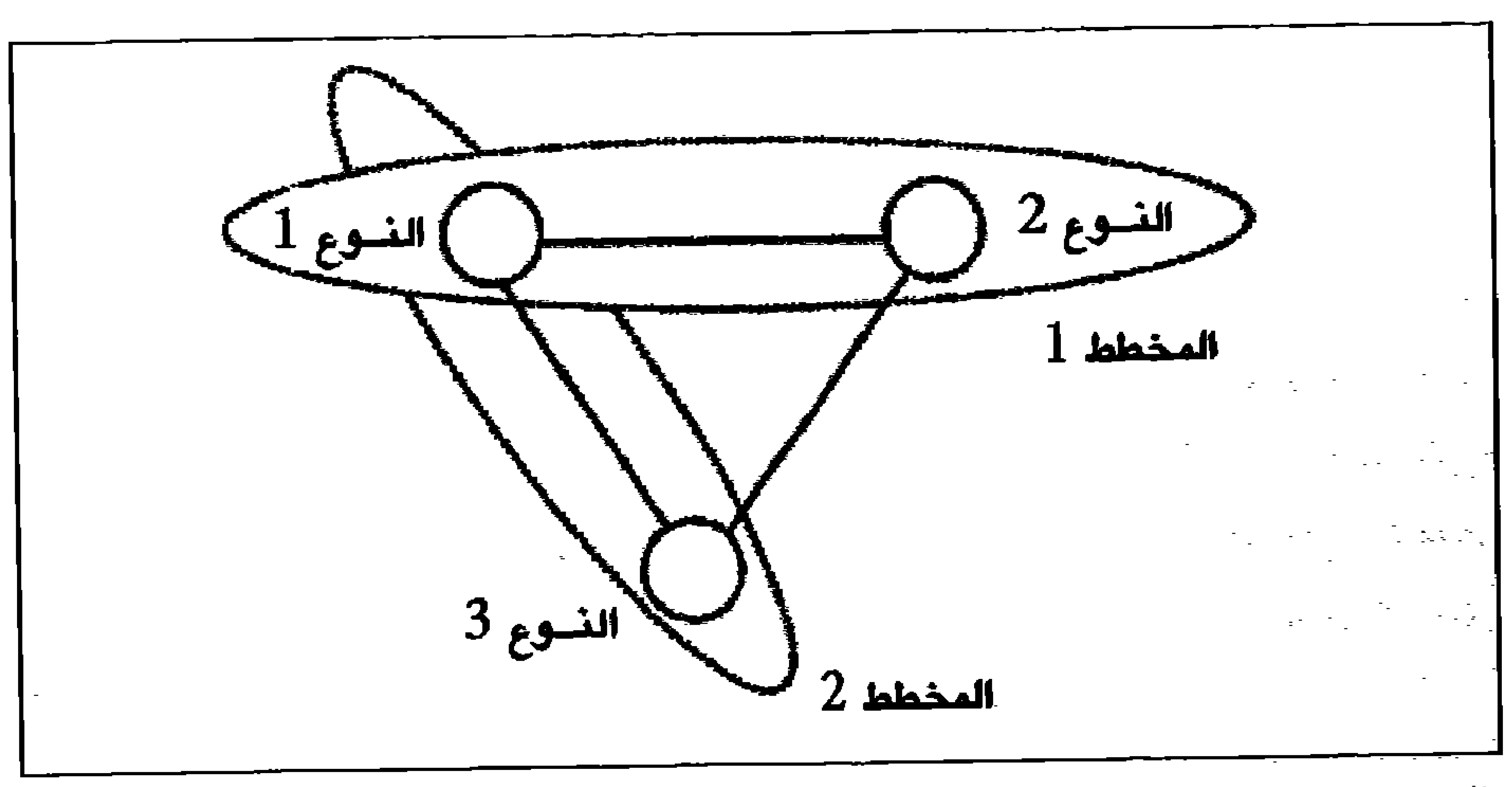
6 ـ 2 نظرة منحازة إلى لغة النمذجة الموحدة

سنبدأ بمراجعة بعض المفاهيم العامة الأساسية للغة النمذجة الموحدة. التصميم هو عبارة عن مجموعة من النماذج التي تصف النظام قيد البناء من منظورات عدة، وكل نموذج هو وصف نظري للنظام أو جزء منه. يُنظّم التصميم في عروض، بحيث يجمع كل عرض النماذج التي تعرض وجهات نظر مترابطة منطقياً عن النظام. العروض النموذجية هي عرض «حالة الاستخدام» الذي يقدم سيناريوهات عن استخدام النظام، والعرض «التشغيلي» المعني باستمرارية السيطرة Control Flow ومشكلات الأداء، والعرض «الفيزيائي» المعني بالربط بين المعدات والبرمجيات، وعرض «التطوير» المعني بتنظيم مكوّنات البرمجية. أما العرض ذو الصلة بأهدافنا فهو العرض «المنطقي» الذي يأخذ بالاعتبار مكوّنات النظام عند مستوى نظري متقدم.

قد يكون النموذج مستقلاً زمنياً (أي ثابت) ويصف بعض عناصر النظام وعلاقاتها: في العرض المنطقي، قد يستخدم النموذج مفاهيم معيّنة في مجال التطبيق، ويعبّر عنها بصورة أنواع (Classes) وارتباطات. على سبيل المثال، سنستفيد من المفاهيم: المدير (Principal) والأساسي (Key) والرسالة (Message) وغيرها. أو قد يكون النموذج ديناميكياً ويصف جزءاً من سلوك النظام من حيث التفاعل بين الكوائن في النموذج الثابت ذي العلاقة (أي إن النموذج الديناميكي يفترض وجود نموذج ثابت). على سبيل المثال، سنأخذ في الحسبان التفاعلات بين العناصر الرئيسة ذات الصلة ببروتوكول الاتصال.

تُبنى النماذج عادة بلغة النمذجة الموحدة وذلك برسم المخططات. من الأهمية بمكان فهم الفرق بين النماذج والمخططات. فالمخطط هو تعبير رسومي لمجموعة من عناصر النموذج، يُعبّر عنها برسم أقواس (تمثل العلاقات) ورؤوس (تمثل عناصر النموذج الأخرى). والنموذج هو بذاته عبارة عن رسم لكن لعناصر ذات دلالة؛ المخطط هو عبارة عن رسم لعناصر مرئية ترتبط بطريقة ذات دلالات. إذاً، في أداة الدعم، النموذج هو عبارة عن بنية عمل بينما المخطط هو بنية عرض. أحياناً، قد لا تكون جميع عناصر النموذج مبينة في المخطط، كما يبدو في الشكل 6-3، فقد يظهر هذا الوضع إذا تم رسم Class2 مع علاقته بـ Class3 في المخطط 2، ومن ثم قُطع من المخطط، لكن لم يُحذف من النموذج. فعلى الرغم من أن المخططات هي طريقة أساسية لإدخال نموذج ما في أداة من أدوات

لغة النمذجة الموحدة، إلا أن هناك طرقاً لإدخال العناصر عبر صناديق الحوار وأدوات تحرير الهيكلية، بحيث تعمل مباشرة في النموذج.

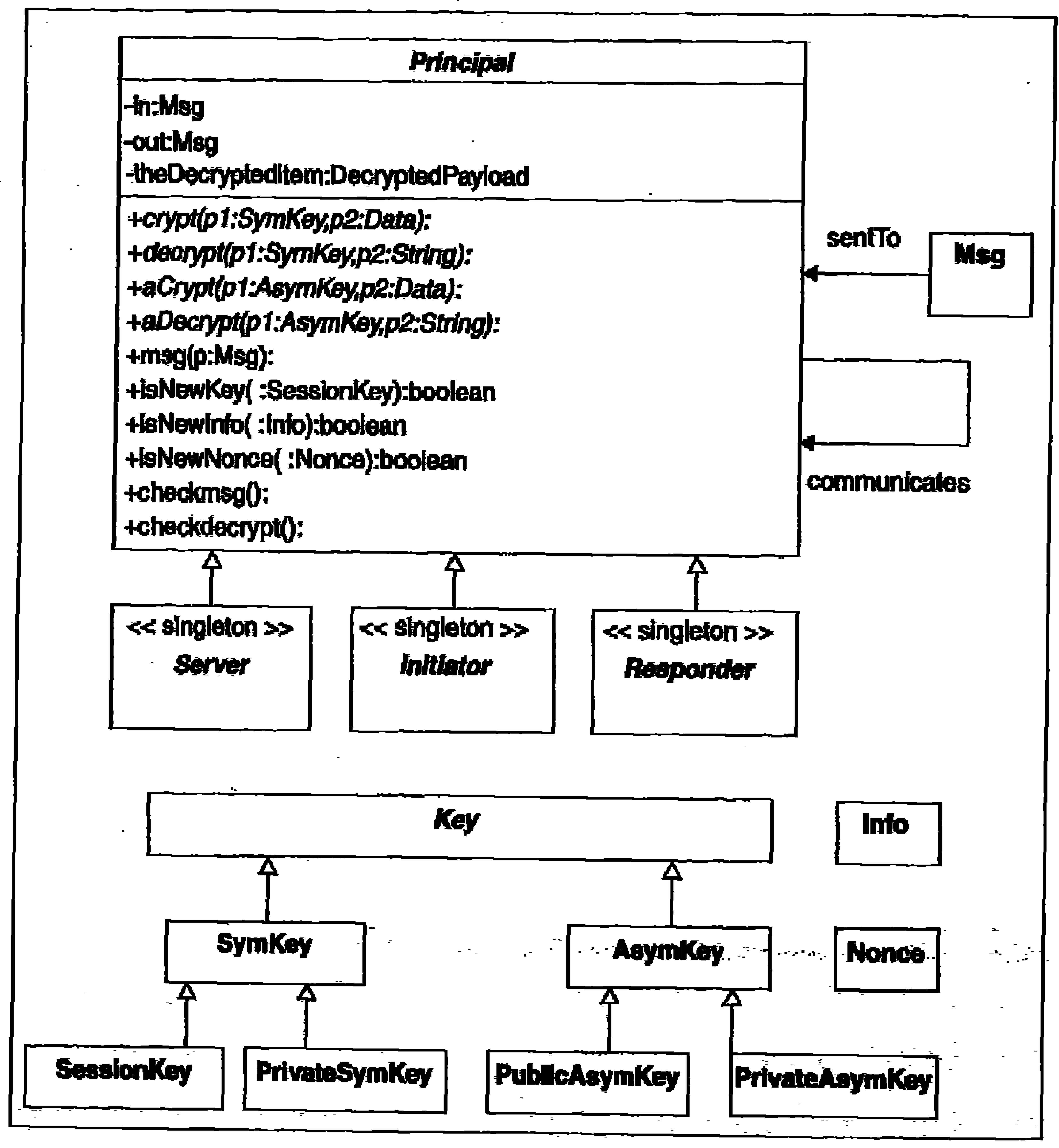


الشكل (6 ـ 3): النماذج والمخططات

في ForLysa، نحدد برتوكول الأمن في عرض لغة النمذجة الموحدة المنطقي، بتوفير نموذجين: نموذج ثابت يصف بنية البروتوكول، ونموذج ديناميكي يصف سلوكه. يُعرض النموذج الثابت كمخطط نوع (Class Diagram)، بينما يعرض النموذج الديناميكي كمخطط تسلسل (Sequence Diagram). نستخدم الإصدار 1.5 من لغة النمذجة الموحدة. في النتيجة، نقدم العناصر التي تلزم لفهم المخططات في القسم التالي بإيجاز. يمكن مطالعة المرجع[9] للحصول على مقدمة سريعة عن لغة النمذجة الموحدة.

يصف مخطط النوع جزءاً من العالم الحقيقي من حيث الكوائن. وبتعبير أكثر دقة، يميز مخطط النوع الكوائن ذات العلاقة بمجال العمل من طريق تصنيفها وعرض بنيتها. النوع هو عبارة عن مجموعة مسمّاة من الكوائن التي لها البنية نفسها والمعطاة كمجموعة من الخصائص ومجموعة من العمليات. في المخطط، يمثل النوع بمستطيل ذي ثلاثة أقسام: يستخدم القسم الأول لاسم النوع، والثاني للخصائص والثالث للعمليات. في بعض الأحيان، قد لا يكون هناك قسم للخصائص أو العمليات، ولا يكون ذلك ضرورياً، كأن يكون فارغاً. فكما هو موضح في الشكل 6-4، جميع مستطيلات الأنواع في الشكل تحتوي على الاسم فقط من دون قسمي الخصائص والعمليات ما عدا النوع Principal

(انتبه، نحن ننظر إلى هذا المخطط في الوقت الحالي من الناحية التركيبية فحسب، وسنتطرق إلى المعنى في القسم التالي). يكون لكلّ خاصية من الخصائص اسم ونوع (كخاصيتي in وout في النوع Principal في الشكل 6-4)، إذ قد يكون الاسم بدائياً أو منطقياً (يحتمل الصواب والخطأ Boolean) أو عبارة عن سلسلة String، أو قد يكون اسم نوع آخر في النموذج. أما العمليات، كالعملية msg، فيكون لكلّ منها اسم وعناصر مطبوعة والقيمة المتوقعة من تنفيذ العملية. يكون للعمليات والخصائص رؤية، قد تكون خاصة بالكائن كما في حالتنا (يعبّر عنها بإضافة إشارة الناقص - في بدايتها) أو عامة (يعبّر عنها بإضافة إشارة الزائد +).



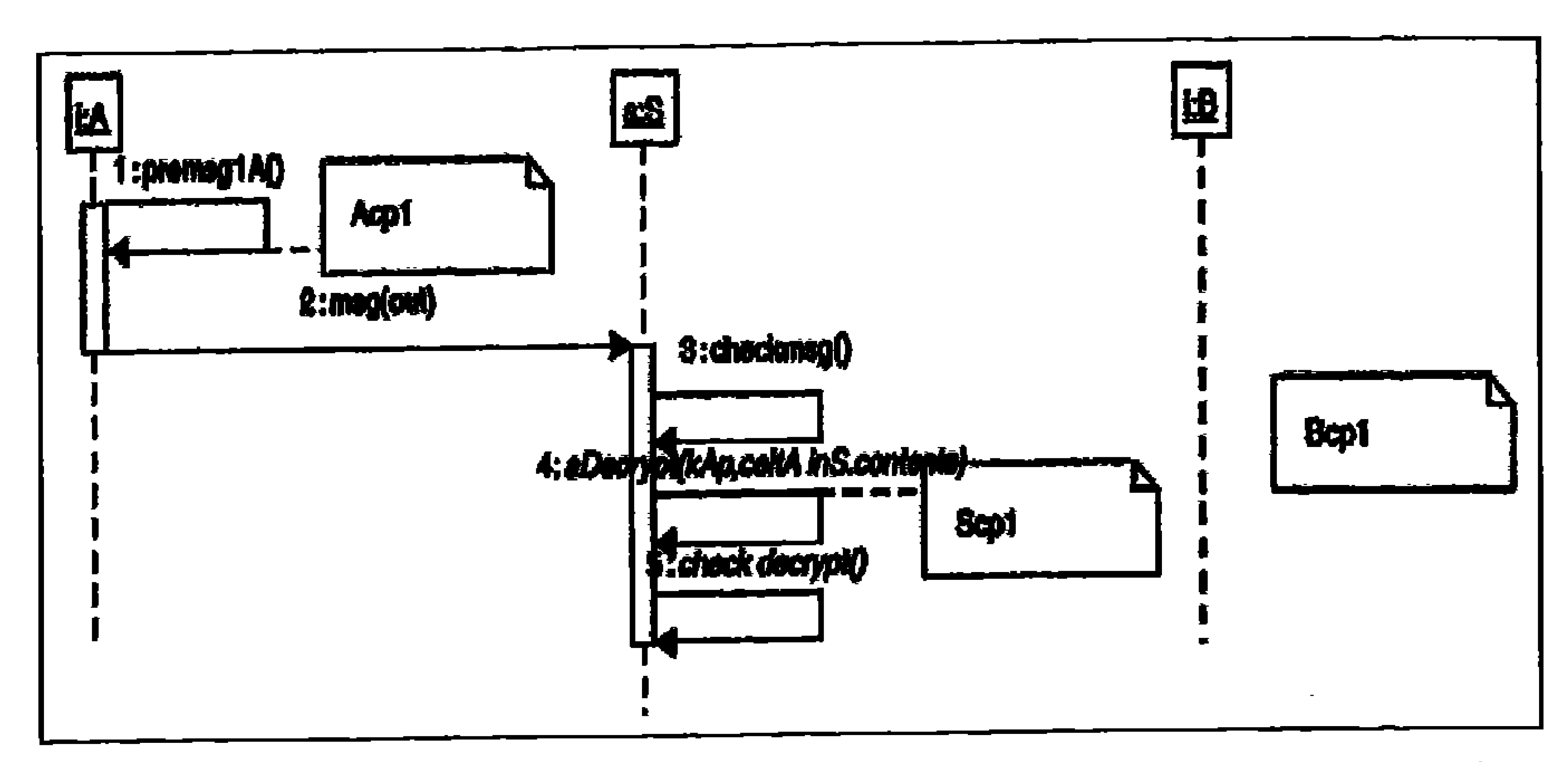
الشكل (6 _ 4) : مخطط النوع Principals

قد تكون العلاقة التي تربط الأنواع من النوع «علاقة Is-A» (ويعبّر عنها بسهم ذي رأس مثلث مفرغ)؛ على سبيل المثال، يبيّن الشكل 6-4 أن الكائن SymKey هو أيضاً Key، أي إن SymKey يخصّص Key أو أن Key يعمم SymKey. إذا كان اسم النوع مكتوباً بالخط المائل فإنه يعبّر عن نوع نظري (بمعنى آخر، نوع ليس له أي كوائن - كالنوع Key في الشكل)، لكنه موجود ليوفر خصائص مشتركة للأنواع التي يعممها. أما العمليات التي تكتب بخط مائل، فهي نظرية أيضاً ويترك تعريفها للأنواع المخصصة.

ثمة نوع آخر من العلاقات موضح في الشكل 6-4. فالخط ذو الرأس المصمت يعبر عن رابطة، ويستخدم للدلالة على علاقات عامة بين الكوائن؛ على سبيل المثال، يمكن أن يتواصل المدير مع مدير آخر، ويمكن إرسال رسالة (من نوع Msg) إلى مدير. كما إن هناك نوعاً آخر من العلاقات موضح في الشكل 6-7، فالخط الذي ينتهي بمعيّن مفرغ يعبر عن «جزء» من العلاقة، أو تجميع. فالكائن الذي في الطرف الآخر من العلاقة يكون «جزءاً» من الكائن الذي يشير إليه المعيّن. على سبيل المثال، في الشكل 6-7، الكائن CryptedPayload له جزء «dest» وهو من النوع SetOfCryptoPoints.

يعطى النموذج السلوكي بواسطة مخطط تسلسل، ومثال عليه المخطط المبيّن في الشكل 6-5. تحدد المخططات التسلسلية جزءاً من سلوك النظام، وذلك بوصف تسلسل محدد للتفاعل بين مجموعة من الكوائن. في هذه المخططات، يسير الزمن من الأعلى إلى الأسفل، بينما يظهر الشركاء ذوو الصلة بالنظام من اليمين إلى اليسار. تحدد هذه المخططات بواسطة الاسم والنوع. على سبيل المثال، الكائن i من النوع A، والكائن s من النوع S. الفترات الزمنية التي تكون فيها هذه الكوائن فاعلة مبيّنة بواسطة مستطيلات مركبة على خط العمر (Lifeline) (الخط المتقطع). أما التفاعلات فهي موضحة بالأسهم ذات الرؤوس السود. كل سهم معنون باسم العملية التي يتم استدعاؤها لتنفيذ التفاعل على الكائن المستهدف. قد تتفاعل الكوائن مع نفسها باستدعاء عملياتها الخاصة، كالكائن preMsg1A في الشكل.

أما المستطيلات ذات الزوايا المثنية في الشكل 6-5 فهي غير مخصصة للمخططات التسلسلية، لكنها مخصصة للاستخدام العام في لغة النمذجة الموحدة لإرفاق الملاحظات لعناصر النموذج. كما سنرى لاحقاً، تؤدي هذه الملاحظات دوراً حاسماً في ForLySa في نمذجة متطلبات المصادقة.



الشكل (6 ـ 5): خطوة بروتوكول نموذجية

6 ـ 3 ForLySA

في هذا القسم، نصف كيفية بناء نماذج البروتوكولات بلغة النمذجة الموحدة، بحيث يمكن تحليلها من حيث خصائص المصادقة في إطار عملي مشروع DEGAS. لهذا الغرض، نعرِّف عدداً من حزم لغة النمذجة الموحدة التي يمكن إعادة استخدامها والتي يجب أن تستخدم في Poseidon، وذلك لجعل نماذج البروتوكول قابلة للتعديل لتحليل DEGAS Choreographer. من ناحية أخرى، يحدد هذا القسم كيفية إنشاء نماذج تتقبلها أداة الاستخلاص في نظام LySA بحيث يكون العاكس (Reflector) قادراً على استخدام التغذية الراجعة عن المنتج.

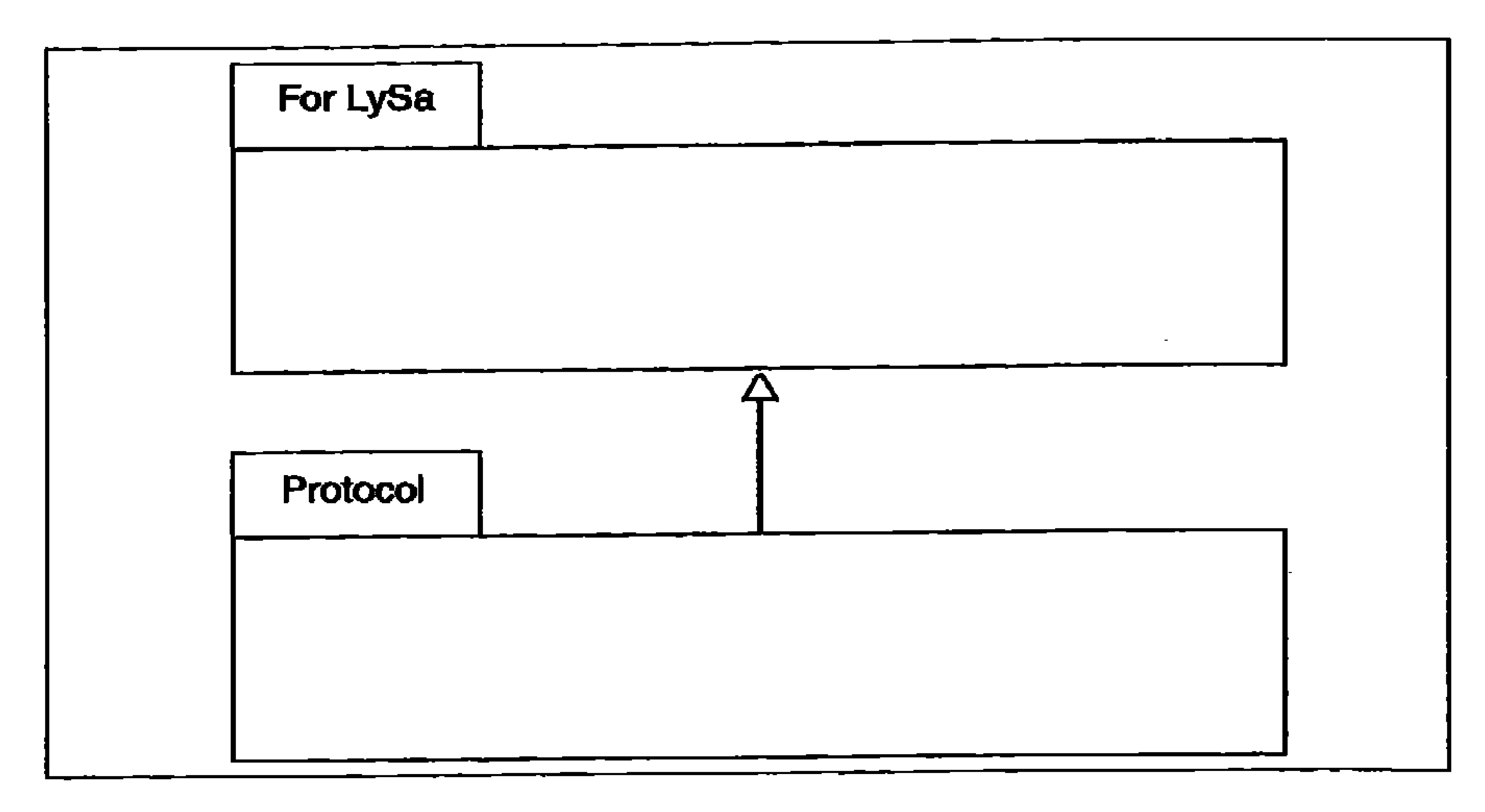
يعتمد التحقق الذي نقوم به على تحليل تدفق السيطرة[3] الذي يتم في LySatool[14]. يبيّن التحليل ما إذا تحققت خصائص المصادقة لجميع محاولات تنفيذ البروتوكول التي أجريت بالتوازي مع عملية الهجوم التعسفي التي تسعى إلى اختراق الاتصال. يخبر التحليل عن جميع الخروقات في خصائص المصادقة باستخدام عنصر مخصص للأخطاء. الثنائي في هذا العنصر (c, d) يعني إذا وجد شيء مشفّر في c فقد تم فكه في d عن طريق كسر خاصية المصادقة. يتم في التحليل حساب overapproximations، يعني أن التحليل قد يبيّن وجود خطأ غير موجود فعلياً. يبيّن المرجع[3] أن ذلك ليس بالمشكلة الكبيرة عملياً.

لتحديد بروتوكول في لغة النمذجة الموحدة بحيث تستطيع أداة

الاستخلاص في ForLySa من توفير المعلومات للمحلل، يجب أن يقوم المصمم وقبل كل شيء بتحديد الاتصالات المقصودة والرسائل المشاركة. تحدد بنية كل نوع من أنواع الرسائل في مخطط رسومي مستقل يتضمن الخصائص الرسومية اللازمة لتحديد خاصية المصادقة. ومن ثم يجب تقديم معلومات كل Principal كمفاتيح الدورة Session keys والتخزين المؤقت، إضافة إلى العمليات اللازمة لبناء الرسائل المحللة بدقة. ثم يعرض المصمم ديناميكيات البروتوكول في مخطط تسلسلي يحدد التنفيذ القانوني للبروتوكول. تقسم كل رسالة يتم تبادلها في البروتوكول إلى ثلاث خطوات: (أ) يجهز المرسل الرسالة، (ب) يتم إرسال الرسالة، (ج) يعالج المستقبِل الرسالة الواردة. يتم وصف كل خطوة بلغة النمذجة الموحدة بواسطة الأسهم في المخطط التسلسلي، بحيث يرتبط كل منها بعملية من العمليات المستهدفة. يتم تحديد كل عملية بشروط مسبقة وشروط لاحقة، مثلاً، لتحديد كيفية فك جزء من الرسالة المشفرة، وما الذي يجب التحقق منه في الرسالة الواردة أو ما الذي يجب تخزينه لاستخدامات المدير لاحقاً. تعرض هذه الشروط على المصمم بدلالات مفاهيم النمذجة في لغة النمذجة الموحدة. تعكس هذه الدلالات الدقة التي تعطيها ترجمة حسابات العملية LySa[3].

أخيراً، تحدد المواقع المذكورة في خصائص المصادقة على شكل ملاحظات مرتبطة بالأسهم في الخطوات من 1 إلى 3 أعلاه، لتوفير الربط اللازم للتغذية الراجعة من عملية التحليل. هذه الملاحظات هي بمثابة موقع تخزين يدعم الإعلام عن الأخطاء الناتجة من عمليات التحليل. إذا كان الخطأ الذي أعلم عنه التحليل في الثنائي (c, d) فإن العاكس سيعدل الملاحظة التي تعرّف c لتتضمن d لتشير إلى وجود خرق في المصادقة التي أشارت إليها عملية التحليل.

كما هو مبيّن في الشكل 6-6، تفترض الأنواع والترابطات التي تحدد سيناريو التحقق أن أداة الاستخلاص توجد في حزمتين. في الحزمة الأولى ForLySa، نقوم بنمذجة مفاهيم عامة كالمدراء والأساسيات والرسائل، وهكذا؛ أما نموذج البروتوكول الفعلي فهو مشتق موسعاً الحزمة الثانية وهي حزمة البروتوكول. تورث بعض الأنواع في البروتوكول من ForLySa بما أنها تخصص المفاهيم العامة للسيناريو الذي تتبعه أداة LySa.



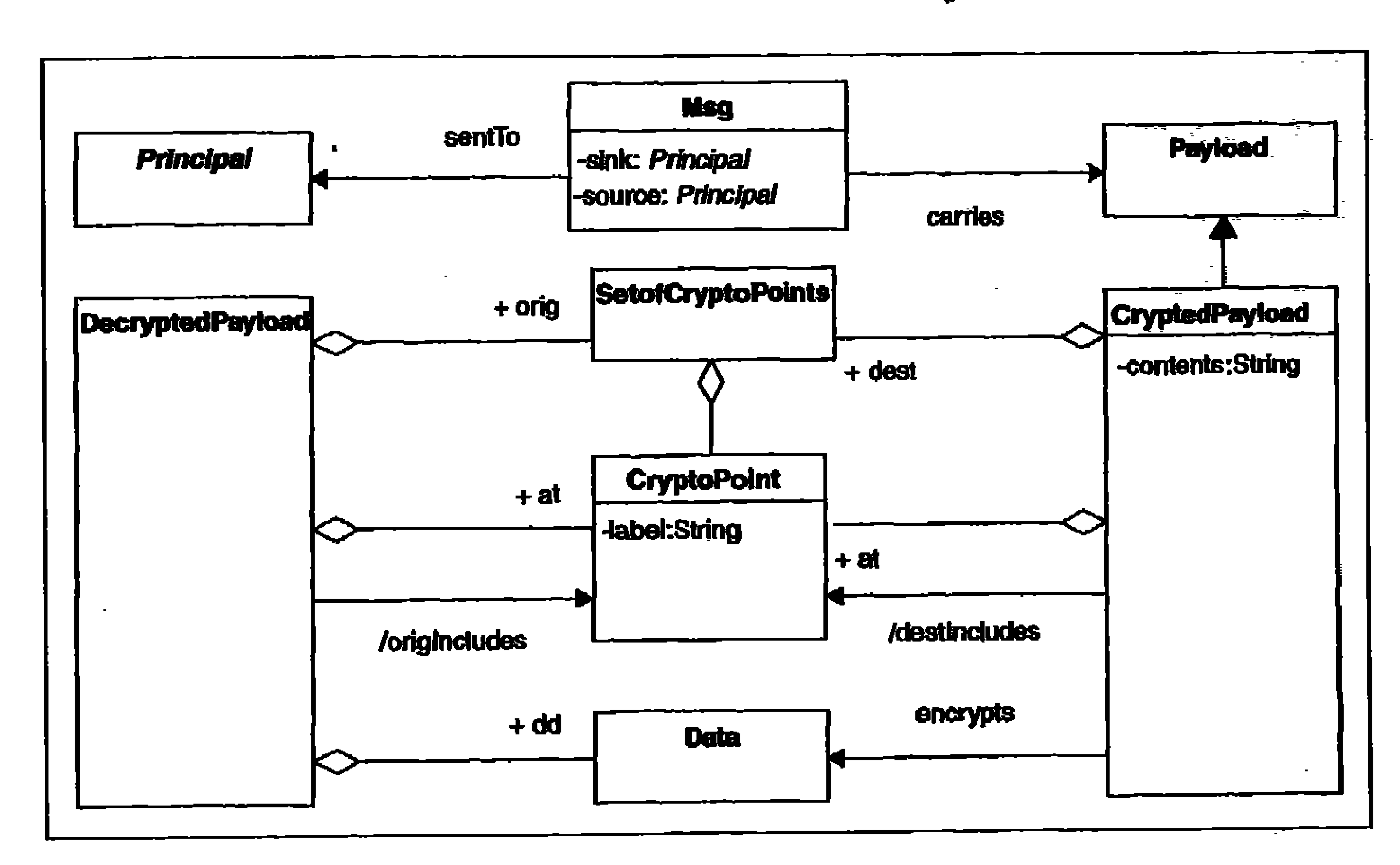
الشكل (6 ـ 6): الهيكلية العامة لـ ForLySa

6 ـ 3 ـ 1 النموذج الثابت

محتويات الحزمة ForLySA مبيّنة في الشكلين 6-4 و6-7، في ما يتعلق بالجوانب العامة والمدراء، وفي ما يتعلق بتفاصيل الرسائل على التوالي. تؤخذ بالحسبان بروتوكولات ذات ثلاثة أطراف، يكون فيها أحد أنواع المدراء كبادئ للبروتوكول، ويكون هناك نوع آخر من المدراء كمستجيب. إضافة إلى ذلك، يوجد خادم يشار إليه أحياناً على أنه طرف ثالث موثوق به أو مركز توزيع أساسي أو سلطة مخوَّلة، وهكذا. في مواصفة البروتوكول، يوجد بادئ واحد ومستجيب واحد وخادم واحد فقط، كما هو مبيّن من خلال الكلمة الدليلية المفردة في المخطط. يتواصل المدراء ويتبادلون الرسائل كما هو محدد في ارتباطات التواصل. للتعبير عن الاتصال، يمكن استدعاء العملية msg () للمدير الذي ترسل له الرسالة. مواصفة هذه العملية هي أنه ينسخ مكوّناته إلى خاصية في المستقبِل. بالنسبة إلى التناسق، ولتسهيل عملية الاستخلاص، نتوقع أن تمرر قيمة الخاصية إلى العملية msg (). ملخص ذلك، عندما يسهم مدير في بروتوكول الاتصال، يجب تتبع الرسالتين: (أ) الرسالة الواردة التي تتركها العملية msg () في المتغير «الوارد» وتطلق المساهمة و(ب) الرسالة الصادرة التي تبنيها العملية في المتغير «الصادر»، ومن ثم ترسلها.

هناك نوعان مختلفان من التشفير المقبول: التشفير المتماثل الذي يستخدم إما مفاتيح خاصة أو مفاتيح دورة، والتشفير غير المتماثل الذي

يستخدم أزواجاً من المفاتيح key pairs عامة أو خاصة. هناك بعض العمليات العمومية التي يمكن استعمالها لبناء الرسائل وفتحها. وتترك هذه العمليات بصورتها النظرية بما إننا ندع اختيار خوارزميات التشفير مفتوحاً لتخصصات أخرى. في الواقع، يتعامل التحليل مع التشفير على أنه عمليات نظرية بحيث لا نحصل على نتائج تحليل دقيقة من تخصيص هذه العمليات أكثر من ذلك. يتم تنفيذ التشفير وفك التشفير من قبل العمليتين () crypt و() decrypt للحالات المتماثلة، أو من قبل العمليتين () aCrypt و() aDecrypt للحالات غير المتناظرة. نفترض أن العمليتين () decrypt و() aDecrypt تتركان نتائج التنفيذ في الخاصية theDecryptedItem. سنناقش نوع هذا المتغير، وهذه العوامل لاحقاً عندما نتناول بنية الرسائل بالتفصيل. يجب أن يتبع تنفيذ هذه العمليات عملية أخرى هي () checkdecrypt، كما هو مبين لاحقاً.



الشكل (6 ـ 7): مخطط النوع للرسائل

يمثّل النوع Info ما يتم إرساله في أثناء التواصل وهو غير مخصص هنا، بينما يمثل النوع Nonce الأعداد المستخدمة مرة واحدة فقط وتعتبر هذه أداة قياسية عند تحديد البروتوكولات.

كما هو مبيّن في الشكل 6-7، تحمل الرسائل (من النوع Msg) قيمة البيانات المرسلة الفعلية Payloads، يمكن أن يكون بعضها CryptedPayloads،

أي تكون البيانات المشفرة ضمن محتوياتها. تنتج قيم CryptedPayloads من عمليات التشفير التي أوضحناها مسبقاً. بشكل مزدوج، تترك عمليات فك التشفير قيماً من نوع DecryptedData في المتغير المحلي المسمى theDecryptedItem، وتتضمن هذه القيم معلومات واضحة في الحقل dd.

لدعم التحليل، قد ترفق نتائج كل عملية تشفير/فك تشفير بعناوين تسمى Cryptopoints، وهي (أ) تعرّف النقطة في البروتوكول بشكل فريد حيثما تم تنفيذ العملية (حسب المثال 6-7، تم التنفيذ في الحقل at) و(ب) تحدد النقاط التي أنشأت فيها البيانات المرسلة الفعلية (النقطة orig في حالة فك التشفير)، أو التي ينوى استخدامها (النقطة dest في حالة التشفير).

إضافة إلى ذلك، ولدعم التحليل دائماً، يتم إرفاق كل رسالة بحقل المصدر ليحتوي على اسم مرسل الرسالة وحقل الهدف الذي يجب أن يحتوي اسم المرسل إليه.

تستخدم العملية checkmsg () لفتح الرسائل والتحقق منها، بينما تستخدم العملية checkdecrypt () لفتح البيانات المرسلة الفعلية المشفرة والتحقق منها. سيتم تحديد الدلالات المحددة لكل استخدام لهذه العمليات في نموذج البروتوكول باستخدام قيود ثابتة وشروط لاحقة. سنناقش تفاصيل ذلك لاحقاً عند التطرق للغة التخصيص.

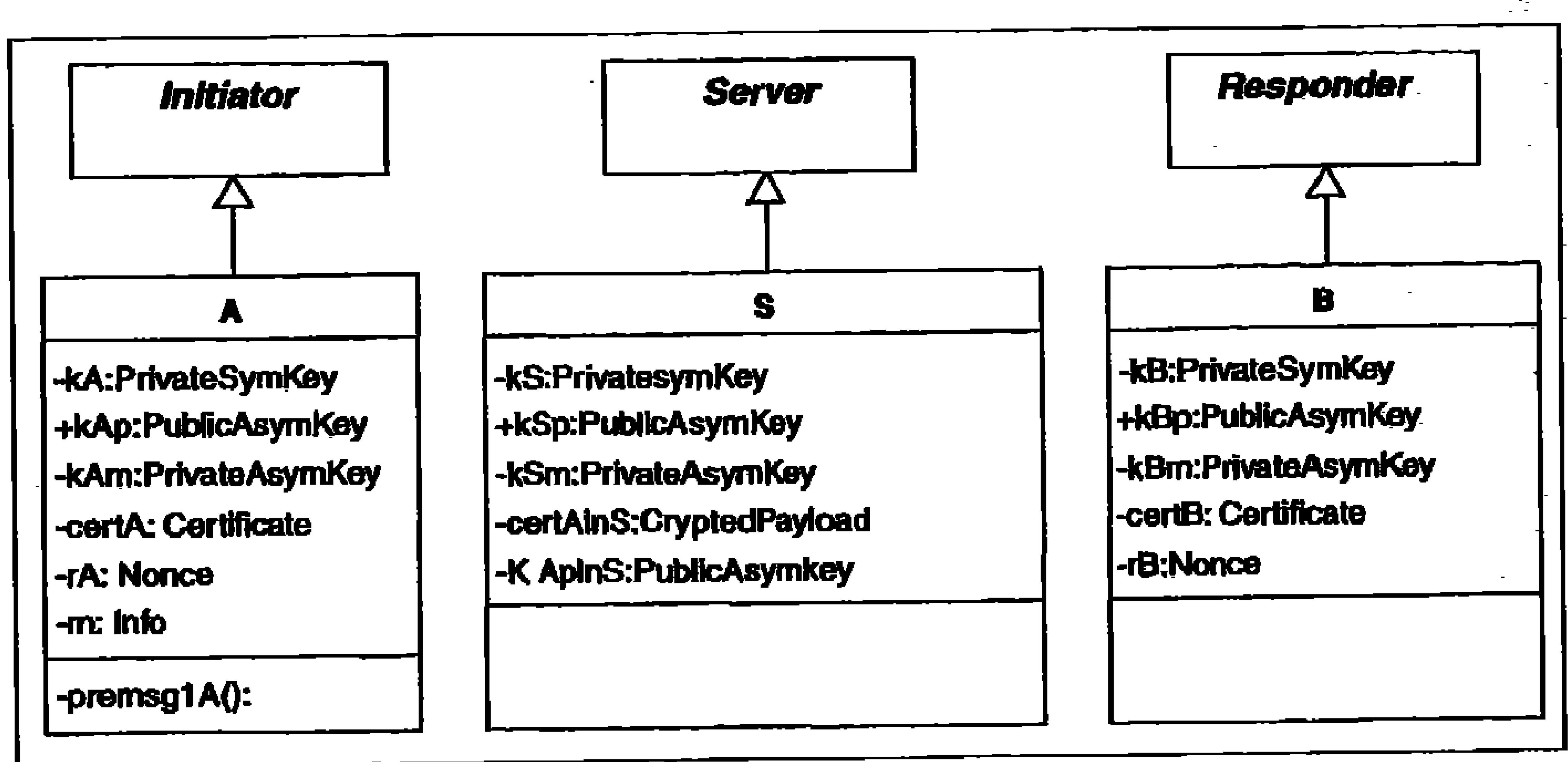
أخيراً، ليس هناك عمليات قياسية لبناء الرسائل الصادرة. يجب أن تعرف هذه العمليات بشكل صريح حتى لو كانت مسمّاة بطرق معيارية، سنناقشها في القسم التالي، ومحددة بواسطة شروط مسبقة ولاحقة. يمكن استخدام أجزاء الرسائل الواردة والمخزنة في المتغيرات المحلية بواسطة العمليات المنفذة مسبقاً/ checkmsg checkdecrypt، إضافة إلى معلومات محددة يعقدها الرئيس في سمات خاصة. طالما أن ذلك يحدث غالباً، فإنه عند تخصيص بروتوكول يحتاج إلى قيمة «حديثة» من نوع ما، يعرف المدير ثلاث عمليات من نوع «isNew» للتعبير عن الحاجة إلى عمل تهيئة صحيحة لبعض المتغيرات. يمكن استخدام هذه الفروض في الشروط المسبقة لعمليات بناء الرسالة.

6 ـ 3 ـ 2 النموذج الديناميكي

لتحديد بروتوكول في منهجيتنا، يلزمنا تحديد أنواع البادئ الفرعية

والمستجيب، والخادم الذي يعرّف المتغير المحلي الذي سيحمل أجزاءً من الرسائل الواردة ومعلومات أخرى معيّنة، إضافة إلى تعريف عمليات معيّنة لبناء الرسائل الصادرة. ثم نحتاج إلى بناء المخطط التسلسلي الذي (أ) يعرض التكامل بين المدراء ذوي العلاقة، (ب) يحمل مواصفات كل عملية، على هيئة قيود على الأسهم.

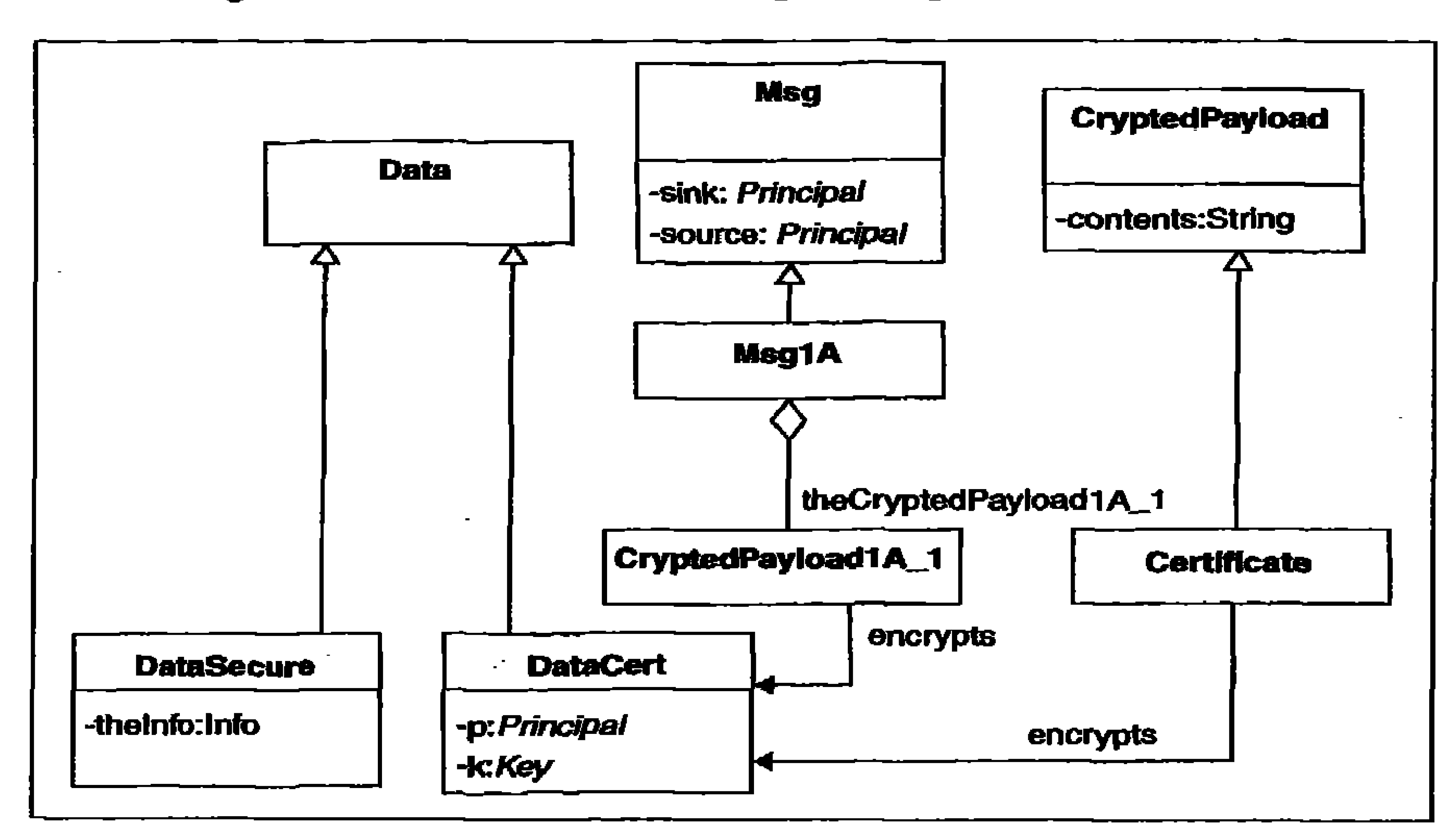
تتضمن حزمة البروتوكول نقطة البداية الأساسية للنشاط بطريقة تعكس سيناريو عملية التحقق الحالية، كما هو موضح في الأشكال 6-8 و6-9 و6-5. أما أسماء البادئ والخادم والمستجيب في الشكل 6-8 فتأتي من الطرق التقليدية غير الرسمية في استخدام البروتوكول.



الشكل (6 ـ 8): مبادئ في سيناريو التحقق

في السيناريو، يتشارك كل مدير مع الخادم لتشفير مفتاح مماثل مع الخادم. تعرف هذه المفاتيح الخاصة بـ kA وkB للبادئ والمستجيب على التوالي. إضافة إلى ذلك، يتم تخصيص زوج مفتاح غير متناظر لكل مدير. تعرف هذه المفاتيح بـ kAp، kAm، kBp، kBm، kSp، kSm للبادئ والمستجيب والخادم على التوالي. يتم اختيار الصفات النهائية لهذه الأسماء لتذكرنا بإشارات الزائد والناقص المستعملة في لغة النمذجة الموحدة للدلالة على الخصائص والعمليات العامة والخاصة على التوالي. في الواقع، يتم الحفاظ على سرية المفاتيح السالبة في أزواج المفاتيح من قبل المدير الذي يمتلكها، بينما تكون المفاتيح الموجبة معرفة لجميع المديرين. وهذا متفق عليه حسب قوانين الوضوحية التابعة للغة البرمجة الموحدة.

أما الخصائص الباقية فهي ليست معيارية لكنها عبارة عن أمثلة بسيطة على ما قد يقدمه مصمم البروتوكول. يتوافق الخيار المعطى هنا مع السيناريو الذي يقوم فيه البادئ بتشفير المعلومات وإرسالها إلى الخادم للحصول على تفويض، حيث يقوم الخادم بدوره بقبولها في certAinS ويعيد التفويض إلى A (أي يعيد المعلومات المستلمة إلى A وهي مشفرة مع المفتاح غير التناظري الخاص kSm)، ثم يقوم A بتخزين التفويض في المتغير certA لاستخدامه مستقبلاً. كما يتوقع هنا أن يقوم البادئ باستخدام رقم خاص rA^(*)Nonce وجزء من المعلومات m. أما المستجيب، فسيكون له تفويض خاص ورقم خاص. أما البنية المثالية للبيانات فهي موضحة في الشكل 6-9 الذي يبيّن هيكلية الرسائل النموذجية Msg1A المستخدمة من قبل البادئ في السيناريو الموضح في ما سبق لإرسال المعلومات اللازمة للحصول على تفويض، وأسس التسمية المتعارف عليها للرسائل.



الشكل (6 ـ 9): بنية بيانات نمطية

(*) الرقم الخاص (Cryptographic Nonce)، هو رقم أو رمز ثنائي (bit) يُستخدم لمرة واحدة فقط، غالباً ما يكون عشوائياً (أو ما يظهر بمظهر العشوائي) يتم إصداره من قِبَل بروتوكول (Authentication Protocol) للتأكد من أن أي اتصال قديم لا يمكن استخدامه في هجوم إعادة الإرسال (Replay Attack) هذه الأرقام الخاصة تختلف في كل مرة يتم فيها تقديم Authentication challenge response code، وكل طلب لعميل له تسلسل رقمي نادر، مما يجعل من هجوم إعادة الإرسال شبه مستحيل. للتأكد من أن الرقم الخاص يتم استخدامه لمرة واحدة فقط يجب أن يكون متغيراً مع الوقت (يحتوي على ختم وقت مناسب ومتغير في جزء من قيمته time-stamp)، أو يتم احتسابه بطريقة معينة ويكون الناتج عدداً معيناً من الـ bits العشوائية لضمان وجود احتمال ضئيل جداً لإعادة احتساب القيمة السابقة العشوائية، وتقليل احتمال تكرار القيم هو متطلب أساسي لاحتساب قيمة هذا الرقم الخاص (Nonce) (المترجم).

أخيراً، بما أن البادئ يحمل عبء إطلاق البروتوكول، فهو يبيّن عملية بناء الرسائل المثالية premsg1A (). هنا، نتبع المتطلبات (أي أن عمليات البناء تحمل أسماء تبدأ بـ premsg) ونبيّن الطريقة المعيارية للتسمية (أي ترقيم العمليات بترتيب حدوثها نفسه ووضع علامات عليها تتضمن اسم المدير الذي يختص بها). تُعطى دلالات هذه العملية وغيرها من العمليات بشروط مسبقة وشروط لاحقة ـ كما أوضحنا مسبقاً ـ وقيود ثابتة في مخطط التسلسل، باستخدام لغة المواصفات المعرفة لاحقاً.

أخيراً، يبيّن الشكل 6 ـ 5 خطوة البروتوكول المثالي، بوصفه نقطة البداية في المخطط التسلسلي. يبني البادئ الرسالة الأولى ويتم التأشير على التشفير عند حدوثه في cryptopoint Acp1، ثم يرسلها إلى الخادم الذي يزود الرسالة ب checkmsg ويفتحها بـ aDecrypt وcheckdecrypt. يبيّن المخطط كلاً من اسم cryptopoints التقليدي المألوف والعناوين ـ أي عن طريق وضع هذا الأخير في ملاحظة ترتبط بالسابق. علاوة على ذلك، يبيّن المخطط الأسماء التقليدية للكوائن Principal في البروتوكول، تحديداً i وj وs للبادئ والمستجيب والخادم على التوالي.

لإكمال مواصفات البروتوكول، يجب أن يعرّف المصمم رسائل جديدة، ويكمل كلّ منها بتحديد الشروط الثابتة المسبقة واللاحقة. سيتم مناقشة مواصفات الرسائل التي تم التقديم لها هنا بعد تقدير لغة المواصفات في القسم التالي.

6 ـ 3 ـ 3 لغة التوصيف

إن بناء الجمل المتبع في اللغة والمستخدم لتحديد العمليات في البروتوكول معطى في الجدول 6-1، حيث تبنينا الاصطلاحات الوصفية القياسية الآتية: الأقواس المربعة للدلالة على عناصر اختيارية، الأقواس المعقوصة لتمثل الحالات التي لم تحدث أو التي تتكرر أكثر من ذلك، والعناصر ذات الخط الغامق تمثل الوحدات الطرفية. نحن نعطي دلالات لغة التوصيف بشكل غير رسمي، كما سيأتي. سندعو الكائن الذي يستهدفه السهم بوجهة العملية وهو الكائن الذي ترتبط معه عملية معيّنة.

المواصفة Spec: تقيّم كل مواصفة بـ Namespace الذي يدمج أسماء عناصر العملية مع Namespace لمخطط التسلسل الذي يحتوي على أسماء الكائن (i وj وs) ويدمجها مع Namespace للوجهة. إذا كان الاسم يشير إلى

كائن، فإنه يمكن الوصول إلى الحقول التي تتبع له أيضاً، وهكذا، بشكل متكرر، بالطريقة المعتادة. علاوة على ذلك، يتضمن تقييم Namespace جميع المفاتيح العامة المعيارية kXp. عندما تكون الوجهة عبارة عن خادم، سيتضمن Namespace جميع المفاتيح الخاصة المناظرة kX، بما إن الخادم يعرفها جميعها حسب هذا السيناريو. المصمم مسؤول عن غياب التعارضات (Clashes)؛ لكن من شأن اصطلاحات التسمية أن تسهّل هذه المهمة.

الثابت Inv: يمكن أن يرتبط الثابت مع Checkmsg للتحقق من الرسائل الواردة أو Checkdecrypt للتحقق من نتائج فك التشفير. الهدف من وجود «ثابت» هنا هو شرط وجوده نفسه على الرسائل في لغة النمذجة الموحدة، إذ يعمل على حجز البروتوكول إن لم يحقق الشرط. من الواضح أنه في النسخة العامة من نظام Poseidon لم توفّر شروط ضمنية للرسائل في مخططات التسلسل في أثناء بناء المصمم Choreographer، لذا توجهنا نحو ذلك التحوّل. القيد الوحيد الذي يمكن فرضه في المتغير هو أن قيم طرفي الشروط متساوية.

الجدول (6 ـ 1)

تركيب لغة التوصيف

```
Spec::= Inv | Pre | Post | Comment
Inv::= Cond { , Cond }
Pre::= TypeRestriction | EncryptedType |
Initialization { & Initialization }
TypeRestriction::= Ide: Type
EncryptedType:: Ide encrypts DataConstructor
Initialization::= isNewKey ( Ide ) | isNewInfo ( Ide ) |
isNewNonce ( Ide )
Post::= [ with TypeRestriction ] Cond { & Cond }
Cond::= Name = Expr
Name::= Ide {. Ide }
Expr::= Name | Fun ( [ Expr { , Expr } ] )
Fun::= crypt | aCrypt | cp | Constructor
Ide::= <any name defined in the namespace of the destination>
Constructor::= SetofCryptpoints | DataConstructor
DataConstructor::= <any data constructor>
Comment::= $$ <string> $$
```

غالباً ما يتم التحقق من المصدر وقائمة انتظار التجميع في كل رسالة مقابل القيمة المتوقعة. يمكن إجراء عمليات تحقق إضافية، لكن ذلك يعتمد على تحديد البروتوكول؛ مثلاً، يجب أن تكون البيانات المرسلة الفعلية الواضحة مساوية لعدد مصادر الرسائل ـ أي إن الطرف المرسل يمكن أن يتحدث مع نفسه فقط. قد يجري تحقق آخر وهو أن يكون المستجيب هو الجهة المقصودة فعلاً. هذه التحققات تجعل البروتوكول أكثر قوة وتمنع أي اختراقات خبيثة في فترة التنفيذ.

الشرط المسبق Pre: يجب أن يرتبط الشرط المسبق مع كل استدعاء للعملية msg ()، وذلك لتحديد نوع الرسالة الحالية الفعلي بواسطة قيد للنوع. كما يجب أن يرتبط شرط مسبق مع كلّ عملية decrypt/aDecrypt، وذلك لتحديد نوع البيانات المشفرة بواسطة بيان خاص بالبيانات المشفرة. أخيراً، تتيح الشروط المسبقة للمصمم التعبير عن حالات البدء Initializations الموجزة في Premsg (راجع ما يرد لاحقاً).

التهيئة Initialization: المعنى المقصود هنا هو أنه قبل تحديد عملية معيّنة، يكون قد خُصص لعناصر هذه المسندات (Predicates) قِيَم جديدة. يمكن استخدام التهيئات لهذا الغرض لتمثل الشرط المسبق لعملية Premsg. في ما يتعلق بالتحليل، فإن غياب التهيئة الملائمة سيقود غالباً إلى كشف المزيد من الأخطاء نظراً إلى أنه يترك بعض القيم غير محمية.

إن اختيار العوامل في التهيئة يعكس مجموعة من الافتراضات في ما يتعلق بالمفاتيح:

- المفاتيح الخاصة، إما أن تكون متماثلة أو غير متماثلة، يمكن استخدامها بحرية في العمليات الخاصة بالمالك، حيث يفترض أنه يمكن تهيئتها قبل بدء البروتوكول.

- المفاتيح العامة غير المتماثلة يمكن أن تستخدم في أي مكان نظراً إلى طبيعتها. لكن قد تقيد بعض البروتوكولات نفسها لاستخدام المفاتيح العامة التي قد تم تبادلها بشكل صريح.

- مفاتيح الدورة يجب أن يتم تهيئتها قبل استخدامها.

يجب أن يتم تهيئة الرقم الخاص وأي معلومات يتم انتاجها وتبادلها في أثناء تنفيذ البروتوكول قبل استخدامها.

الشرط اللاحق .Post: يستخدم الشرط اللاحق لتحديد تأثير عملية معيّنة في حالة الوجهة المستهدفة. الشرط اللاحق للعملية msg هو قياسي، تحديداً «in = p,» وهذا يعني أنه يتم نسخ الرسالة المرسلة (قيمة العنصر p المخصصة هي قيمة الخاصية «out» للمرسل) إلى الخاصية «in» للمستقبل.

في حالة الشرط المسبق Premsg، نستخدم عادة عنصراً اختيارياً، وهو هنا عنصر فتح نطاق السجل، لا تحتاج الحقول إلى أن تكون مسبوقة بالمعرّف، في الطرف الأيسر من الشروط التي تتبعها. القيد هنا هو «out:MsgM» إذا كانت العملية هي premsgM[1]. وهذا يستلزم أن تكون أسماء البيانات المرسلة الفعلية الخاصة بـ MsgM قابلة للاستخدام للدلالة على الحقول التي يجب تخصيصها.

التأثير المقصود من هذه العملية مبيّن في الشروط المبيّنة الآتية.

الشرط Cond: عند استخدام الشرط في ثابت، فإنه يعبّر عن حالة تحقق، كما هو مبيّن أعلاه. عند استخدام الشرط المسبق، فإنه يعرّف قيمة الخاصية أو أحد الحقول كنتيجة لتنفيذ العملية التي يحددها؛ يدل الاسم في الجانب الأيسر على العنصر الذي يأخذ القيمة المعرفة في العبارة الموجودة في الطرف الأيمن.

الاسم Name: اسم يشير إلى متغير. يستخدم رمز النقطة المعيارية للدلالة على الوصولية إلى حقول الكائن: على سبيل المثال، يشير N.I إلى أن المتغير ممثل بـ I في موقع الاسم الخاص بالكائن بدلالة N.

التعبير Expr: يشير التعبير إلى متغير. يستخدم رمز النقطة المعيارية للدلالة على الوصول إلى حقول الكائن: على سبيل المثال، يشير N.I إلى أن القيمة ممثلة ب I في موقع الاسم الخاص بالكائن بدلالة N.

المعرّف Ide: يشير المعرّف إلى قيمة أو متغير اعتماداً على السياق ويقييم كما يأتي:

---

(1) يجب أن يظهر القيد نفسه كشرط مسبق في عملية msg اللاحقة.

- تقييم عوامل العملية بالنسبة إلى العناصر المناظرة لها كما هو محدد في مخطط التسلسل.

- تقييم المعرّفات في مخطط التسلسل بالنسبة إلى الكوائن المناظرة لها.

- تشير خصائص كائن (بما في ذلك الكوائن المعيارية) إلى الكائن أو القيمة المناظرة حسب نوعها.

- تقييم أسماء المفاتيح الأساسية حسب المفتاح المناظر لها.

تقييم عناصر العملية بطريقة مشابهة في الموقع المخصص للاسم Namespace عند منشأ السهم.

Fun: تتضمن العمليات التي يمكن قبولها إجراءات التشفير وإنشاء البيانات التي سيتم تشفيرها وإنشاء **Cryptopoints** ومجموعات **Cryptopoints**. باتباع الجافا، يكون مع المنشِئ اسم نوع الكوائن التي يجب بناؤها. يجب أن تتوافق العناصر مع حقول النوع في العدد والترتيب. أما عناوين **Cryptopoints** فتكون ضمن مصرح بها ضمنياً، ويستخدم **cp** كاختصر ل **CryptoPoint**.

قبل أن نتناول مثالاً، علينا أن نذكر بعض العمليات المعيارية واصطلاحات التسمية المتبعة فيها.

الرسائل **Messages**: يجب أن تكون الرسائل من النوع **MsgM**، حيث **M** هو رمز فريد لبنية رسالة معطاة. يجب أن يتم تسمية البيانات المرسلة الفعلية بـ **thePayloadY** و**theCryptedPayloadZK**، حيث **Y** و**Z** يضيفان إلى الرمز **M** رقماً يبيّن ترتيبه في الرسالة.

البيانات **Data**: يجب أن تكون البيانات المشفرة من نوع **DataM**، حيث **M** هو رمز فريد لبنية معطاة من البيانات.

المدراء **Principals**: يجب أن يكون لكل مدير عدد من عمليات **PremsgM** بعدد الرسائل التي يرسلها، حيث **M** هو نوع الرسالة المرسلة نفسه. يجب أن يتم تسمية عناصر كل عملية (إن وجدت) على النحو **p1**، **p2**، . . . حسب ترتيب ظهورها. أما كوائن المدير فيجب أن تسمى **i**، **j**، **s**.

Cryptopoints: يجب أن يتم تسميتها بطريقة معيارية: **TcpN**، حيث **T** هو نوع المالك، و**N** هو عدد تصاعدي.

الجدول (6 ـ 2)
نموذج سردي

```
1:premsg1A () $$ A -> S: {A,kAp}:kSp $$
<postcondition>: with out:Msg1A source = i & sink = s &
theCryptedPayload1A_1.contents = acrypt (kSp,DataCert (i,kAp) ) &
theCryptedPayload1A_1.dest = SetofCryptpoint (Scp1) &
theCryptedPayload1A_1.at = cp (Acp1)
2:msg (out)
<precondition>: out: Msg1A
<postcondition>: in = p
3:checkmsg () $$ A -> S: {A,kAp}:kSp $$
<invariant>: in.source=i, in.sink=s
<postcondition>: certAinS.contents =
in.theCryptedPayload1A_1.contents
4:aDecrypt (kAp,certAinS.contents)
<precondition>: certAinS encrypts DataCert
<postcondition>: theDecryptedItem.at = cp (Scp1) &
theDecryptedItem.orig = SetofCryptpoint (Acp)
5:checkdecrypt ()
<invariant>: theDecryptedItem.dd.p=i
<postcondition>: kApInS = theDecryptedItem.dd.k
```

6 ـ 3 ـ 4 مثال على مواصفة العملية

لمناقشة استخدام لغة التوصيف، نستغل المخطط الموضح في الشكل 6-1 والسرد المبيّن في الجدول 6-2، الذي يعرض جميع الشروط المرتبطة بالعمليات في النموذج، ويمكن إنشاؤها أوتوماتيكياً على هيئة المصمم Choreographer من نموذج لغة النمذجة الموحدة الموضحة في ما سبق(*).

يمكن التعبير عن الجزء المخصص للبروتوكول الذي تناولناه هنا بالأسلوب غير الرسمي المستخدم لمناقشة القضايا المتعلقة بالتشفير على الصورة

A-> S: {A, kAp}: kSp

(*) خاصية المصمم Choregrapher هذه مفيدة جداً عند تتبع أخطاء النموذج وتصحيحها. في الواقع، ارتبطت هذه الشروط بالأسهم في المخطط من خلال أطراف الإدخال/ الإخراج في محرر النموذج، ويمكن فحصها على حدة فقط، بحيث يكون من الصعب تكون تصوّر شامل عن المواصفة (المترجم).

للتعبير عن أن البادئ A يرسل رسالة إلى الخادم S، وهو يتكون من الزوج، <اسم البادئ ومفتاح البادئ العام> مشفراً مع مفتاح الخادم العام. إن حقيقة كون التشفير غير المتماثل المستخدم يبقى ضمنياً، بناءً على اسم المفتاح. وتبيّن أن وصف هذه الخطوة كخطوة شائعة في العملات التي تبني وتستلم الرسائل هي طريقة مفيدة جداً لتوثيق البروتوكول. إضافة إلى اسم ورقم كل عملية، تعرض كل مدخل في الجدول 6-2 يعرض المعلومات الآتية عند توفرها، وذلك لكل سهم في المخطط التسلسلي: التعليق من حقل «التوثيق» متضمناً برمز الدولار مزدوجاً وحقول القيود مرتبطة بنوعها.

تعمل العملية الأولى في هذه الخطوة من البروتوكول (premsg1A) على بناء الرسالة التي يكون نوعها Msg1A كما هو مصرح به في قيود النوع في الشروط اللاحقة وهي تتطلب تعريف الحقول الخمسة الآتية:

- مصدر الرسالة، ويحمل هذا الحقل اسم كائن البادئ، i.
- قائمة انتظار تجميع الرسائل، ويحمل هذا الحقل اسم كائن الخادم، s.
- حقل CryptedPayload المعياري ومحتوياته، ويحمل هذا الحقل نتيجة فك تشفير غير المتماثل مع مفتاح عام للخادم a وبيانات نمط DataCert (بناءً على ما هو مصرح به في الشكل 6-9 من قبل رابطة التشفير). أما حقول البيانات المشفرة فهي، كما هو معرّف أعلاه، اسم البادئ والمفتاح العام.
- حقل CryptedPayload المعياري، والاتجاه، ويحمل هذا الحقل Cryptopoint مرتبطة مع مكان فك التشفير المقصود، الذي يحدده المصمم بـ Scp1 حسبما ورد في الشكل 6-5.
- حقل CryptedPayload المعياري ويحمل هذا الحقل Acp1، Cryptopoint المرتبطة بهذه العملية في الشكل 6-5.

المدخل الثاني في الجدول 6-2 هو معياري، ويوفر جميع المعلومات اللازمة لتحديد النقل من الخارج (في البادئ) إلى الداخل في الخادم.

أما المدخل الثالث فهو يكرر التعليقات؛ وهذا أمر مفيد للمصمم الذي قد يلقي نظرة على شيفرة LySa المنشأة. وحيث إن Premsg و Checkmsg لم تعد متجاورة، فإن التعليقات التي يخبر عنها في شيفرة LySa تساعد في ربط القطع

بعضها ببعض. تالياً، يعبر الثابت عن أنه قد تم التحقق من حقول الرسالة المعيارية والمصدر وقائمة انتظار التجميع، وذلك للتأكد من أنها ما يتوقعه المصمم.

إضافة إلى ذلك، يؤكد الشرط اللاحق على أن الجزء المعلوماتي من CryptedPayload للرسالة الواردة يخزن في المتغير المحلي certAinS لإجراء المزيد من المعالجة. أما الحقول الأخرى dest وat فهي ليست ذات صلة هنا.

أما المدخلان الأخيران، فيجب مراعاتهما معاً وذلك أنها يتعاونان في فتح Payload المشفر والتحقق من محتوياته ومن أجزاء التخزين كمرجع مستقبلي. يجب أن يعلن الشرط المسبق للعملية رقم 4 عن النمط الفعلي للحقل dd للمتغير المحلي الذي يقبل نتيجة فك التشفير theDecryptedItem. أما الشرط اللاحق للعملية نفسها فيعرّف الحقول الأخرى، وهي:

- الحقل الذي يعنون نقطة فك التشفير at التي تأخذ القيمة Scp1 حسب الشكل 6-5.
- الحقل الذي ينص على المنشأ المتوقع، أي موقع التشفير، ويأخذ القيمة Acp1.

يعرّف المدخل الأخير حالات التحقق من نتيجة فك التشفير (نتطلب أن يكون مدير التحويل هو البادئ نفسه) وأن يكون المفتاح مخزناً محلياً لاستخدامه لاحقاً من قبل الخادم لبناء التخويل للبادئ.

6 ـ 3 ـ 5 الانعكاس

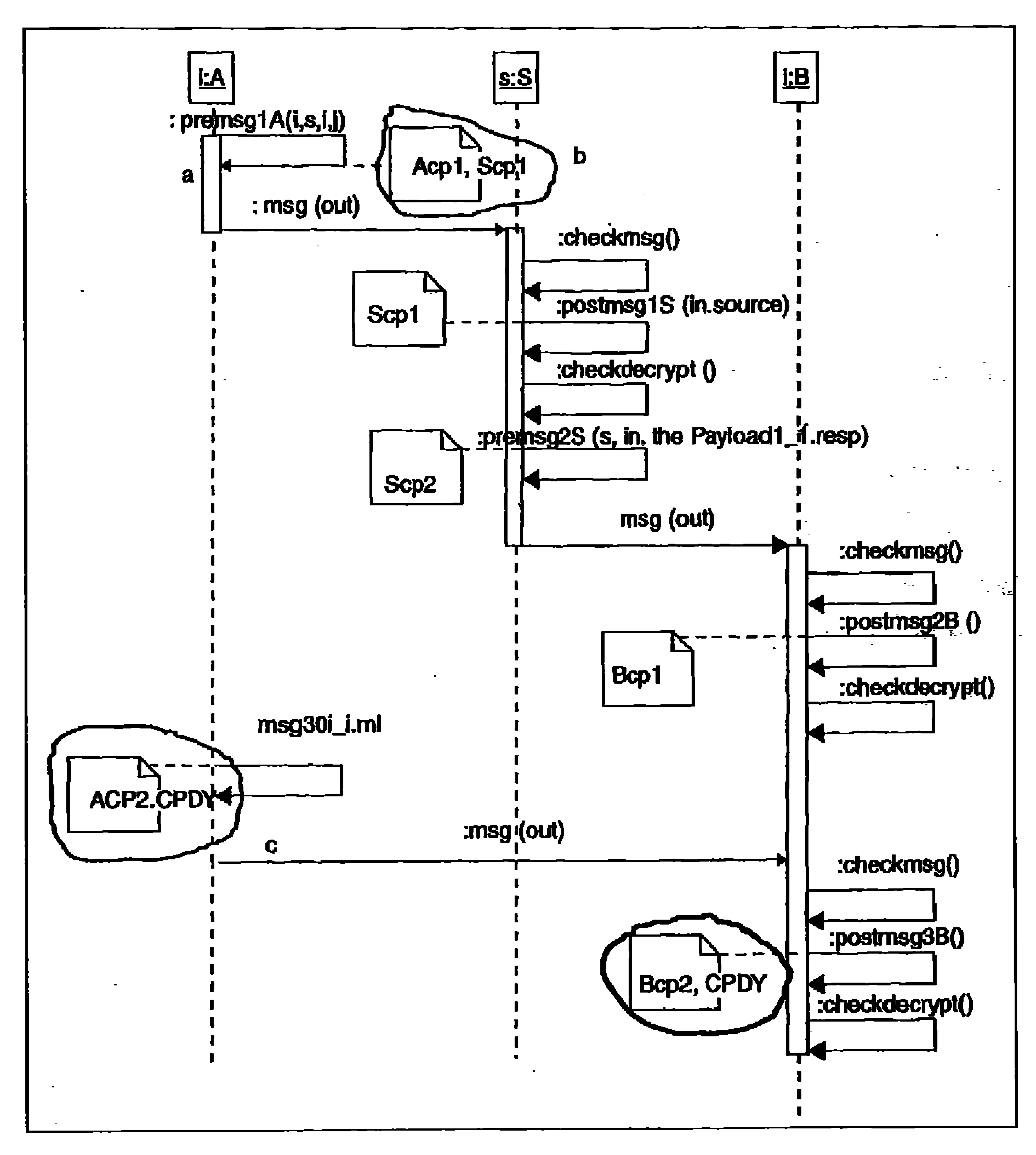
نناقش الآن باقتضاب كيفية عرض نتائج التحليل على المصمم. يبيّن الشكل 6-10 نتائج انعكاس الإصدار المتدفق من برتوكول «الضفدع ذي الفم الواسع»(*) الشهير. يصف البروتوكول التغير الأساسي بين مديرين A وB خلال خادم موثوق به. هناك ثلاث خطوات للبروتوكول:

1. يرسل المدير A رسالة إلى الخادم تتضمن اسم B ومفتاح الدورة session الجديدة K_{AB}، وتكون هذه المعلومات مشفرة مع المفتاح العام K_{AS}.

(*) يمكن مطالعة تفاصيل عن هذا البروتوكول بالرجوع إلى الموقع الإلكتروني: <http://en.wikipedia.org/wiki/Wide_Mouth_Frog-protocol>.

2. يقوم الخادم بفك تشفير المعلومات ويرسل اسم A والمفتاح الجديد K_{AB} إلى إلى B، وتكون هذه المعلومات مشفرة ضمن المفتاح العام K_{BS} الخاص بـ B.

3. يرسل المدير A رسالة إلى B مشفرة ضمن المفتاح K_{AB}.



الشكل (6 ـ 10): مثال على مخرجات عملية التحليل.

في أثناء تحليل هذا البروتوكول، تعمل الأداة على إيجاد الأخطاء المحاطة بدوائر في الشكل 6 ـ 10. على سبيل المثال، إن فك التشفير عند Bcp2 قد يفك تشفير الرسائل الواردة من مهاجم (يرمز لها بـ CPDY) بدلاً من

فك تشفير الرسالة الواردة من Acp2 فقط كما يجب. بناء على هذه الأخطاء، قد يتمكن المصمم من تحديد المشكلة وتعديل وصف البروتوكول في لغة النمذجة الموحدة، ويعيد تنفيذ التحليل إلى أن يضمن المحلل عدم وجود أي أخطاء في البروتوكول.

6 ـ 3 ـ 6 الخبرة

إضافة إلى الأمثلة الصغيرة المستخدمة لاختبار الأدوات، تحققت خبرات واسعة في مجال منهجية التحقق في مشروع DEGAS مع العديد من دراسات الحالة. كان ثمة اثنتان منها قدمها شركاء صناعيون.

دراسة الحالة الأولى هي عبارة عن لعبة فيديو يمكن أن يلعبها عدد كبير من اللاعبين في الوقت نفسه في سياق شامل وبيئة مفتوحة. هناك خادم رئيس مهمته الرئيسة التنسيق فحسب، حيث تتم التفاعلات بين اللاعبين مباشرة بين أجهزة الهواتف الخلوية على أساس نظير ـ نظير، من خلال ترابط البيانات أو الرسائل القصيرة. من الضرورة بمكان أن يتم التواصل بين اللاعبين، وبين كل لاعب والخادم بطريقة آمنة لتجنب إمكانية أن يغير المهاجمون متغيرات اللاعبين والكوائن التي يجمعونها في أثناء اللعب. نظراً إلى طبيعة النظير ـ النظير الخاصة بالمشروع، يجب على البروتوكول «الأمان» أن يقلل من تدخل الخادم.

أما دراسة الحالة الثانية، فتتضمن خدمة تعمل من خلال الويب لتمكين تنفيذ الأعمال الإلكترونية الجزئية بناءً على المصادقة نظير ـ نظير ونموذج التواصل. والهدف من ذلك توفير آلية تجارة إلكترونية نظير ـ نظير بسيطة لمجموعة المشترين والبائعين، عن طريق عرض تسهيلات الأعمال من خلال الويب للبائعين الذين ليس لديهم مصادر لتطوير حلول امتلاك، كما أنها تعرض وصولية سهلة لمعلومات المنتجات والخدمات للبائعين. تتم معالجة التحويلات المالية بواسطة خدمات بنكية. يجب أن يكون التواصل ممكناً من خلال اتصالات الإنترنت السلكية ومن خلال الأجهزة الخلوية باستخدام بروتوكولات كبروتوكول التطبيقات اللاسلكية WAP. يمكن تقليل متطلبات الأمان لدراسة الحالة هذه لتأسيس موثوقية للرسائل، التي تعتبر خاصية الأمان الرئيسة في مشروع DEGAS.

في كلتا الحالتين، من المفضل استخدام التشفير المتناسق للتبادل نظير ـ نظير بين الأجهزة الخلوية، طالما أن الحمل الحاسوبي لها أصغر مما في التشفير غير المتناسق. لكن يجب أن تكون المفاتيح التي سيتم استخدامها محمية من الدخلاء الماكرين. لذا، ابتكر بروتوكول مكوّن من جزئين: أحدهما يتضمن مفاتيح عامة موزعة، والثاني يتفق فيه زوج من اللاعبين على استخدام مفتاح دورة. تبيّن التحليلات أنه يجب استخدام وسائل أخرى لتأسيس موثوقية للرسائل؛ يجب أن يفوض كلا اللاعبين بعضهما بعضاً بهدف منع أي جاسوس من إدخال المفتاح الخاص به بدلاً من المفتاح العام لأحد اللاعبين.

البروتوكول الناتج يدعى SecureSend، وله مخطط تسلسل ذو 38 سهماً، وقد تم تحليله بنجاح في مصمم Choreographer. هذا ويمكن الرجوع لمزيد من التفاصيل إلى المراجع[15] و[17].

6 ـ 4 الاستنتاجات

إن الهدف الإجمالي لعملنا شبيه بعض الشيء بهدف أطر العمل ك Casper[13]، وCAPSL[6]، وCVS[8] وAVISS[1]. تهدف أطر العمل هذه جميعاً إلى تزويد المطوّرين ببروتوكولات الأمان ببيّنات متقدمة لأدوات التحليل؛ لكن وخلافاً لمنهجنا، فأطر العمل هذه ترتكز على الملاحظات المخصصة. من ناحية، قد يؤدي ذلك إلى وصف أكثر إحكاماً للبرتوكولات من وصفنا؛ لكن من ناحية أخرى، لدينا كافة مزايا استخدام لغة النمذجة ذات الأغراض العامة. فمن الناحية التقنية، المعلومات الموجودة في أوصاف البروتوكول في أطر العمل المذكورة أعلاه شبيهة بالمعلومات الملتقطة في مخططات تسلسل الرسائل خاصتنا. إضافة إلى ذلك، نجد بعض التشابهات على وجه الخصوص في نظام Casper الذي له تشكيل تحليلي للهدف باستخدام عمليات تفاضل وتكامل. عملية الفلترة التي تتم في Casper أبسط من عمليتنا لأن لغتها المتقدمة مصممة بحيث تتضمن مباشرة عمليات معالجة للتعبيرات الحسابية في أماكن مريحة.

ثمة جهود مهمة تتشارك في مشروع DEGAS تركز على لغة النمذجة الموحدة وهي مركزية في UMLsec. وهذا عبارة عن ملف بلغة النمذجة الموحدة للتعبير عن معلومات ذات صلة بالأمان ضمن المخططات في

مواصفات النظام، وفي المنهجية المرتبطة بتطوير نظام آمن[11]. يتيح UMLsec للمصمم التعبير عن متطلبات الأمان متكررة الحدوث كالتبادل المباشر والأمان/السرية وتدفق معلومات الأمان ورابط التواصل الآمن. ثمة قواعد للتحقق من صحة نموذج من حيث متطلبات الأمان، استناداً إلى دلالات أساسية لجزء لغة النمذجة الموحدة المستخدم. يتيح لنا هذه الأساس الدلالي على وجه الخصوص التحقق ممّا إذا كانت القيود المرتبطة بقوالب لغة النمذجة الموحدة النمطية متحققة بمواصفات معيّنة. العمل جارٍ لتوفير دعم تحليلي أوتوماتيكي، بمنهجية شبيهة بمنهجية نظام DEGAS. برأينا، يوفر نظام ForLySa طريقة أكثر حدسية للتعبير عن متطلبات المصادقة أقل مركزية في UMLsec: إذ يجب أن يكون تقييم جدوى الاندماج بين المنهجيتين أمراً مجدياً.

إن الخبرة المكتسبة في العمل على دراسات حالة في مشروع DEGAS تقترح أن تقديم بروتوكول التحليل الذي رأيناه أعلاه مفصل جداً حتى لا يبدو مستخدماً بكفاءة عند استثمار البروتوكول في تصميم تطبيق ما. يجب أن يكون المصمم قادراً على استخدام بروتوكول في تطبيق ما بسهولة حالما يتم تفويضه. أساسياً، لا يرغب المرء برسم أكثر من سهم لكلّ خطوة في البروتوكول، عند تصميم تطبيق، أو قد يرغب في تحديد بعض المعلومات المتبادلة فحسب باستخدام البروتوكول. أما أفضل طريقة لربط هذه العروض المختصرة والكاملة للتحليل فقد تركناها لمزيد من الأبحاث.

من القضايا الأخرى المفتوحة واحدة ترتبط بتنفيذ البروتوكول. فمن المعروف أن عملية التنفيذ يمكن أن تؤدي إلى عيوب، حتى لو بدأنا من مواصفة صحيحة مثبتة. الآن وفي حالتنا، تتضمن المواصفة المستخدمة لتحليل البروتوكول معلومات كافية لدعم التنفيذ، والهدف المتمثل في اشتقاق التنفيذ الصحيح أوتوماتيكياً يستحق السعي وراءه.

الحلّ المستخدم للإبلاغ عن معلومات الخلل عند مستوى لغة النمذجة الموحدة هو حل معتدل بعض الشيء ويمكن جعله ممكناً عن طريق حقيقة أن المخطط يحضّر بحيث يوفر تثبيتاً للإدخال: أي الملاحظات المرفقة مع الأسهم. على كلٍّ، يبدو هذا الاعتدال تبادلاً جيداً إلى أن يتم تبادل المستحقات في مخطط لغة النمذجة الموحدة المعياري المتطورة[18]. في الواقع، لا يبدو أن بذل الكثير من الجهود في حل عروض رسومية محددة لمخطط بيئة نمذجة

واحدة أمر مجدٍ. فحالما يكون المعيار متوفراً، سيكون مجدياً تصميم خوارزمية لتحديث المخطط بإضافة نتائج التحليل بطريقة أكثر مرونة، والتي ستكون لها شرعية عامة ضمن المعايير.

الاستنتاج، نتذكر أن العمل الذي ورد هنا كذلك الذي ورد في التحليل الكمي هو أيضاً مضمّن في المصمم Choreographer وهو نوع من الهجوم قصير الأمد للمشكلة الموضحة في المقدمة ـ أي أنه يتم صقل تأثيرها السلبي في عملية التطوير لمنهجيات واسعة من خلال واجهة لغة النمذجة الموحدة.

على المدى الطويل، يمكن حل المشكلة بواسطة هجومات متقاربة تتضمن جميع اللاعبين في اللعبة: يجب أن يتم الاستثمار أكثر في البحث والتطوير وفي تدريب العاملين، بحيث لا يعود السلوك الشكلي مثبطاً لهمة المحترفين، كما يجب أن يتعلم الأكاديميون المهندسين للبدء بها ويجب أن يستهدف البحث تكاملاً أساسياً لنظريات متنوعة تختص بالمواصفات الأساسية.

شكر وعرفان

أفاد هذا العمل من العديد من الحوارات التي دارت مع العاملين في مشروع DEGAS، في أثناء الاجتماع بهم في بيزا وترينتو وإدينبرة. نوجّه شكراً خاصاً لكلٍّ من تشيارا بوديه (Chiara Bodei)، بيرباولو ديغانو (Pierpaolo Degano)، وميخائيل بوتشولتز (Mikael Buchholtz) وذلك لتعريفنا بنظم LySa's arcana؛ نشكر أيضاً ميكاييلا فاساري (Michela Vasari) ومونيكا ميدل (Monika Maidl) لبصيرتيهما النافذة في بروتوكول دراسة حالة، ونشكر لازا بيرون (Lara Perrone) وسيمون سيمبريني (Simone Semprini) ودانييلا بيتشيايا (Daniele Picciaia) لبرمجة أداة الاستخلاص، وفال هاينيل (Val Haenel) لدمج أداة الاستخلاص مع Choreographer وبناء العاكس.

المراجع

1. A. Armando [et al.]. «The AVISS security protocol analysis tool.» paper presented at: *Proceedings of the 14th International Conference on Computer Aided Verification (CAV 2002)*, LNCS 2404, pp. 349-353. Berlin: Springer, 2002.

2. K. Beck [et al.]. Manifesto for Agile Software Development. Available at <http://www.agilemanifesto.org>.

3. C. Bodei [et al.]. «Automatic validation of protocol narration.» Paper presented at: *Proceedings of the 16th Computer Security Foundations Workshop (CSFW 2003)*. New York: IEEE Computer Society Press, 2003, pp. 126-140.

4. M. Buchholtz [et al.]. «End-to-end integrated security and performance analysis on the DEGAS Choreographer platform.» paper presented at: *Proceedings of Formal Methods 2005*, LNCS 3582. Berlin: Springer, 2005, pp. 286-301.

5. M. Buchholtz [et al.]. «For-LySa: UML for authentication analysis.» paper presented at: *Proceedings of the 2nd International Workshop on Global Computing, (GC'04)*, LNCS 3267. Berlin: Springer, 2004, pp. 92-105.

6. G. Denker, J. Millen, and H. Rue. The CAPSL integrated protocol environment. Technical Report SRI-CLS-2000-02, SRI International, 2000.

7. Design Environments for Global ApplicationS: DEGAS, project IST-2001-32072, Information Society Technologies programme of the European Commission, Future and Emerging Technologies. Available at <http://www.omnys.it/degas/main.html>.

8. A. Durante, R. Focardi, and R. Gorrieri. «A compiler for analyzing cryptographic protocols using non-interference.» *ACM Transactions on Software Engineering and Methodology*: vol. 9, no. 4, 2000, pp. 488-528.

9. M. Fowler. *UML Distilled: A Brief Guide to the Standard Object Modelling Language*. New York: Pearson Education, 2004.

10. J. Hillston. *A Compositional Approach to Performance Modelling*. New York: Cambridge University Press, 1996.

11. J. Jurjens. *Secure Systems Development with UML*. Berlin: Springer, 2005.

12. P. Kruchten. *The Rational Unified Process: An Introduction*. Reading, MA: Addison-Wesley, 1998.

13. G. Lowe. Casper: «A Compiler for the Analysis of Security Protocols.» *Journal of Computer Security*: vol. 6, no. 1, 1998, pp. 53-84.

14. Mikael Buchholtz. LySa-A process calculus. Web site hosted by Informatics and Mathematical Modeling at the Technical University of Denmark, April 2004. <http://www2.imm.dtu.dk/cs_LySa/lysatool/>.

15. C. Montangero. ForLysa User's Guide. DEGAS Document WP3-UNI-PI-I02-Int-001, 18/2/05. Available at <http://www.di.unipi.it/~monta/ForLySa/ForLySaManual.pdf>.

16. J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language*. 2nd ed. Reading, MA: Addison-Wesley, 2005.

17. D. Spunton [et al.]. Case Studies. DEGAS Deliverable 26, March 8, 2005. Available at <http://www.omnys.it/degas/documents.html#D26>.

18. Unified Modeling Language: Diagram Interchange version 2.0. 2005. Available at <http://www.omg.org/docs/ptc/05-06-04.pdf>.

7

تطوير تطبيقات الويب الحديثة

مهدي جزائري (Mehdi Jazayeri)
وسيدريك ميسناج (Cédric Mesnage)
وجيفري روز (Jeffry Rose)

7 ـ 1 المقدمة

عرف العالم شبكة الويب العالمية عام 1994، التي كانت تهدف إلى إتاحة الوصول إلى المعلومات من أي مصدر بطريقة سهلة ومتسقة. هذا وقد تم ذلك في المنظمة الأوروبية للبحث النووي CERN في جنيف في سويسرا، وقد كانت هذه الشبكة هدف الفيزيائيين وعلماء آخرين ممن ينتجون كميات هائلة من المعلومات والوثائق ويحتاجون إلى مشاركتها مع غيرهم من العلماء. تم تبني النصوص التشعبية كطريقة سهلة لإعطاء وصولية للوثائق وربط بعضها ببعض. بروتوكول نقل النصوص التشعبية HTTP مصمم لإتاحة المجال أمام أحد أجهزة الحاسوب ـ حاسوب تابع ـ من طلب البيانات والوثائق من جهاز حاسوب آخر ـ الحاسوب الخادم ـ بحيث تكون تلك الوثيقة متاحة لمستخدمي الحاسوب التابع. بهذه الطريقة، استعرضت شبكة الويب العالمية كمستودع واسع للمعلومات التي توفّر وصولية بعدد كبير من المستخدمين. لقد تطورت الويب كمستودع ثابت إلى حد بعيد بمرور الوقت. أما الآن، فالويب عبارة عن منصة معقدة تعرض طائفة ضخمة من الأدوات والمكوّنات لمطوّري التطبيقات. ثمة جيل جديد من التطبيقات يوفّر للمستخدمين فرصاً للتواصل والتشارك وتحديث إمكانيات تطبيقاتهم. تدعم التطبيقات الأعمال الصغيرة أو المجتمعات الصغيرة من المستخدمين، إضافة إلى أعمال الشركات الكبيرة.

يستمر الويب في التطور بوتيرة متسارعة. هناك العديد من الأفكار عن التوجهات التي ستسود في المستقبل. ثمة مفهوم الشمول الكامل يراعي الاتجاهات الحديثة كصياغة أساس الويب الإصدار 2.0. وبمقارنتها بتطبيقات الويب التقليدية، كانت تطبيقات الإصدارين 1.0 و2.0 ديناميكية أكثر وكانت تستدعي مشاركة المستخدم، وكانت مستجيبة لطلبات المستخدم كما هو الحال في تطبيقات سطح المكتب. قد يكون أفضل وصف للويب إصدار 2.0 هو ذلك الـذي قـدمـه O'Reilly <http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html>. يقارن هذا المقال التطبيقات والتقنيات من الويب 1.0 والويب 2.0. من الأمثلة التي يطرحها البعض: ويكيبيديا مقارنة بالموسوعة البريطانية، حيث ربط فيها المؤسسون مساعدة المستخدمين بإنشاء بيانات رسم، ثم بعدئذ رسم علامات تمييز واضحة بين منتجي ومستهلكي البيانات. مثال آخر على ذلك، النشر في المواقع الإلكترونية مقارنة بالتدوين التي تختلف بطرق العرض التي تتيح مشاركة المستخدمين. كمثال ثالث، يمكننا احترام استخدام محركات البحث للعثور على المواقع اللإلكترونية خلافاً للطرق السابقة التي كانت تحتم على المستخدم تذكر أو تحزير عناوين المواقع الإلكترونية. ما الذي جعل النسخة 2.0 من الويب ممكنة، وما هو التالي؟

في هذا الفصل، ننظر إلى حالة المهارات والمفاهيم المهمة التي تقع ضمنها ونستحث تطوير الويب 2.0، وأخيراً، التوجهات القادمة في تطبيقات الويب، ودعم البنية التحتية الضمنية.

7 ـ 2 أساسيات الويب

على الرغم من التطورات الهائلة التي حدثت في العقد الأخير، ظلت المبادئ الأساسية التي قامت عليها الشبكة العالمية ثابتة. من الناحية الهيكلية، ترتكز شبكة الويب العالمية على الحوسبة التابع ـ الخادم، حيث تخزن الخوادم الوثائق، بينما تعمل الأجهزة التابعة على الوصول إلى هذه الوثائق. هذا وقد يكون جهاز الحاسوب تابعاً أو خادماً في أوقات مختلفة. قدمت شبكة الويب العالمية ثلاثة مفاهيم أساسية عن الحوسبة التابع ـ الخادم، وهي: طريقة تسمية الوثائق والإشارة إليها (محددات مواقع المصادر الموحدة URL)، لغة لكتابة الوثائق التي تحتوي على بيانات وروابط لوثائق أخرى (لغة ترميز النصوص التشعبية HTML) وطريقة لتواصل أجهزة الخوادم

والتوابع مع بعضها بعضاً (بروتوكول نقل النصوص التشعبية HTTP).

محدد مواقع المصادر الموحدة URL: النظام المسمى هو المكوّن الأساسي للنظم الحاسوبية، مخصص للنظم الموزعة. يصف النظام المسمى طريقة تسمية الكوائن بحيث يمكن تعريف هذه الكوائن وتحديد مواقعها. اعتماداً على خصائص النظام المسمى، يمكن البحث عن الكوائن على أساس أسمائها الدقيقة فقط، أو على أساس مواصفاتها. على سبيل المثال، قد يرغب أحدهم في إخبار النظام ب «ابحث عن الوثيقة التي كتبها آلان تيرنينغ وموضوعها الاختبار الذكي». تهدف طريقة التسمية في شبكة الويب العالمية إلى تحديد الكوائن المخزنة في الحواسيب المرتبطة بشبكة الإنترنت تحديداً فريداً. ترتكز طريقة التسمية على محددات مواقع المصادر الموحدة URL وهي أسماء مركبة تحدد الحاسوب (أي عنوان بروتوكول الإنترنت) والوثائق في نظام الملفات في ذلك الحاسوب. تعرّف محددات مواقع المصادر الموحدة الآن كمعيار في المذكرة IETF RFC 1630.

لغة ترميز النصوص التشعبية HTML: تكتب الوثائق على شبكة الويب بلغة ترميز النصوص التشعبية. تتضمن وثائق لغة ترميز النصوص التشعبية HTML محتويات للعرض وتعليمات منسّقة تُعلِم المتصفح عن كيفية عرض محتويات الوثيقة وروابط الوثائق الأخرى. تطورت لغة ترميز النصوص التشعبية HTML مع تطور متصفحات الإنترنت لتحقيق عروض مرئية وتوحيد المقاييس بشكل أفضل.

مبدئياً، عرضت لغة ترميز النصوص التشعبية HTML كلغة لإصدار التعليمات للمتصفحات لتحديد ما يتم عرضه للمستخدمين. لكن وحيث إن عدد الوثائق المكتوبة بلغة النمذجة الموحدة يزداد، وحيث إن هناك العديد من التطبيقات التي بدأت بإنشاء وثائق بلغة ترميز النصوص التشعبية، أصبحت معالجة الحاسوب لهذه الوثائق أمراً في غاية الأهمية. أما لغة الترميز الموسعة XML فقد أُنشئت لتوحيد تعريف لغات الترميز المخصصة الأخرى. كما أن لغة ترميز النصوص التشعبية الموسعة XHTML هي عبارة عن لغة الترميز الموسعة XML المتوافقة مع لغة ترميز النصوص التشعبية HTML، التي أصبحت متغيراً مسيطراً للغة ترميز النصوص التشعبية.

حالياً، تعمل مجموعة عمل تقنية تطبيقات الويب ذات النصوص التشعبية

(www.whatwg.org) على تعريف المسار التطوري للغة ترميز النصوص التشعبية والتوفيق بين التناقضات بين لغة ترميز النصوص التشعبية HTML ولغة ترميز النصوص التشعبية الممتدة XHTML. هذا وتعمل مجموعات أخرى ك W2C على لغة ترميز النصوص التشعبية الممتدة كمعيار.

بروتوكول نقل النصوص التشعبية HTTP: هو بروتوكول الاتصال لشبكة الويب (راجع RFC 2616)، وهذا البرتوكول يحدد ثماني عمليات أساسية: خيارات OPTIONS، اجلب GET، تصدّر HEAD، أرسل POST، ضع PUT، احذف DELETE، تتبع TRACE، اربط CONNECT. الأكثر استخداماً بين هذه العمليات هما اجلب GET وأعلن POST. فالعملية GET تسترجع البيانات المرتبطة بمحدد مواقع المصادر الموحدة URL من محدد معطى. أما العملية POST فترسل البيانات إلى البرنامج الذي يتلقى محدد مواقع المصادر الموحدة المطلوب.

أثبتت هذه المفاهيم البسيطة أنها فاعلة وقوية وقد كان ذلك مفاجئاً. وجد مطوّرو التطبيقات طرقاً مبتكرة لاستخدام محدد مواقع المصادر الموحدة URL لتسمية أشياء متنوعة، ولا يقتصر ذلك على الوثائق. على سبيل المثال، من الأفكار التي طرحت مبكراً لاستخدام محدد مواقع المصادر الموحدة URL لتسمية برنامج يرغب بتنفيذه على الجهاز الخادم، الذي سينتج منه مخرجات يتم إعادتها إلى التابع. شبيه ذلك أن الوثائق لا تستخدم لاحتواء المعلومات التي سيتم عرضها من خلال متصفح الإنترنت فحسب، بل أنها أيضاً تحتوي على مخطوطات الشيفرة البرمجية التي يجب تنفيذها. ثمة منظومة كاملة من اللغات التي أنشئت لكتابة الشيفرة البرمجية التي يجب تنفيذها من قبل المتصفح (مثال ذلك، JavaScript) أو على الخادم (مثال ذلك، PHP). تستفيد تطبيقات الويب من تنوع اللغات وأنماط التصميم لدمج قدرات الخوادم والتوابع.

7 ـ 3 هندسة البرمجيات وتطبيقات الويب

أبصرت هندسة البرمجيات النور عام 1968 كرؤية لترويض الصعوبات التي اعترضت تطوير البرمجيات في ستينيات القرن العشرين في مشاريع تطوير البرمجيات الكبيرة المبكرة. فقد أصبح واضحاً أن البرمجيات الكبيرة المتجانسة التي كانت النوع الوحيد من البرمجيات، التي كانت قيد الإنشاء، كانت ذات قيود كثيرة. لم يكن ممكناً توسيع تلك البرمجيات في ذلك الوقت

لتتجاوز حجماً معيّناً، كما لم يكن بالإمكان صيانتها، وبالتالي كان من النادر أن تتوافق هذه البرمجيات مع توقعات العملاء. كانت هندسة البرمجيات واعدة بمنهجية نظامية لبناء البرمجيات بناءً على تصاميم الوحدات ومكوّنات البرمجية المعيارية. أصبحت رؤية هندسة البرمجيات هذه حقيقة واقعة بمرور السنين.

مرّ تطوير تطبيقات الويب بتاريخ شبيه بهندسة البرمجيات، لكنه سار على وتيرة أسرع. فقد تكوّنت تطبيقات الويب الأولى من العديد من النصوص البرمجية المبعثرة في العديد من الملفات، التي افتقدت الهيكلية المنظمة. وقد يكون سبب ذلك أن تطبيقات الويب موزعة بطبيعتها، لكن فكرة المكوّنات المفردة التي يمكن جمعها لتكوّن تطبيقات أصبحت حقيقة، وهي أسهل بكثير في تطبيقات الويب. تستخدم تطبيقات الويب المكوّنات المعيارية كوحدات الدفع أو التسجيل. في هذا القسم، نتناول العديد من مظاهر تطبيقات الويب المرتبطة بقضايا هندسة البرمجيات ارتباطاً وثيقاً.

7 ـ 3 ـ 1 ثابت ـ ديناميكي ـ نشط

تمّ بناء الويب في مراحله المبكرة بواسطة مجموعة من الوثائق التي كان بالإمكان ربطها بحرية مع بعضها البعض. كانت تلك الوثائق عبارة عن ملفات نصية بسيطة ذات محتويات ثابتة وروابط ثابتة تربط الصفحات. بمرور الوقت، أدرك الناس أن هذه الوثائق النصية المرسلة من جهاز خادم الويب إلى متصفح الويب يمكن أن يتم إنشاؤها بسهولة بواسطة برنامج. وقاد ذلك إلى ظهور تطبيقات «واجهة البوابة المشتركة CGI»، حيث يشير محدد المواقع URL إلى نصوص صغيرة أو برامج مترجمة كبيرة يمكن تشغيلها من خلال الخادم لإنشاء صفحات ويب ديناميكية. كان برنامج واجهة البوابة المشتركة الذي تم تشغيله من خلال الخادم يستخلص البيانات من قاعدة البيانات. أصبح ذلك مرهقاً للعديد من التطبيقات لأن وضع جميع محتويات موقع إلكتروني وتهيئتها على هيئة شيفرة برمجية لتطبيق ما قد يصبح أمراً مملاً ويصعب إدارته كما يعلم مهندسو البرمجيات. أدى هذا الإدراك إلى نموذج هجين نوعاً ما، حيث كانت لغة البرمجة PHP ذات تأثير مبكرٍ في ذلك. باستخدام PHP كانت الهيكلية النموذجية تقتضي إنشاء صفحات ويب تتضمن وضع كمية صغيرة من الشيفرة البرمجية في نصوص الصفحات مباشرة. في الوقت الذي يرسل فيه طلب، يتم

تنفيذ الشيفرة البرمجية وإدخال النتيجة في الوثيقة الحالية. أدى ذلك إلى مرونة كبيرة جداً وسهولة في إعادة استخدام مكوّنات الصفحة؛ لكن وبمرور الوقت، أدرك الناس صعوبة إجراء تعديلات على الشيفرة البرمجية عندما كانت منتشرة في جميع صفحات الموقع. أدى ذلك إلى إجراء المزيد من التنقيح على النموذج لتقارب ما هو شائع اليوم في عالم تطبيقات الويب، حيث إن المكوّنات المركزية في التطبيقات تحكم الوصول إلى البيانات، بينما تكون الشيفرة البرمجية الموزعة في صفحات HTML محدودة بما يتم عرضه للمستخدمين فقط.

7 ـ 3 ـ 2 نمط تصميم متحكم عرض النموذج

من وجهة نظر هندسة البرمجيات، تبدو متعلقات العديد من تطبيقات الويب متشابهة: فهي جميعها لها واجهة استخدام، ويرتكز استخدامها على متصفح الإنترنت وتتفاعل مع المستخدمين وتتحكم بأكبر كمية ممكنة من البيانات، ويتم تخزينها في الخادم. استغرق الأمر عدة سنوات قبل أن يدرك مطوّرو الويب أن نمط متحكم عرض النموذج MVC المعروف في هندسة البرمجيات ينطبق كما هو الحال في تطبيقات الويب المماثلة.

تمّ تقديم متحكّم عرض النموذج MVC[22] لأول مرة في Xerox PARC عام 1978، واستخدم للمرة الأولى في تطبيق Smalltalk قبل 20 عاماً من انتشار استخدام الويب، وقبل 30 سنة من بدء استخدامها في أطر عمل تطوير تطبيقات الويب كنمط بنيوي جوهري. إن الهدف من نمط تصميم متحكم عرض النموذج حسبما يفيد مخترعوه هو جسر الهوّة بين النمط الذهبي للمستخدم والنموذج الرقمي المتواجد في الحاسوب. في الأصل، كان المتحكم عبارة عن نظام مكوّن من أربع مواد: النموذج (Model) والعرض (View) والمتحكم (Controller) والمحرر (Editor). نوضح متحكم عرض النموذج في سياق تطبيقات الويب. النموذج هو عبارة عن مجموعة من الأنواع كائنية التوجه، تتفاعل مع مستودع البيانات (الذي يكون قاعدة بيانات في الأغلب) المتحكم فيتضمن منطق التطبيقات أو العمليات، أما العرض فهو مخطوط لتجهيز البيانات من النموذج لإنشاء العرض. يعمل المتحكم على إنشاء العروض والاستجابة للاستعلامات التي تنتج من العروض.

7 ـ 3 ـ 3 أطر عمل تطبيقات الويب

استغرق الأمر عقوداً حتى تصل هندسة البرمجيات إلى أطر عمل تطوير تطبيقات يمكن استخدامها، بحيث تدعم هذه الأطر تطوير تطبيقات معقدة بناءً على مكوّنات وتصاميم متشابهة. الفكرة الرئيسة هي توفير هذه المكوّنات في إطار عملي قابل للتهيئة والتوسيع. تتوفّر الآن منهجيات هندسة الويب القائمة على المكوّنات بوفرة[19, 11] ما يدعم التراكيب كائنية التوجه[10]، واستكشاف الاعتمادية القائمة على المظاهر والسياق[3]، والبحث عن أنماط باستخدام لغة النمذجة الموحدة[2]. مؤخراً، ظهر إطار عملي قوي وسريع في العديد من لغات البرمجة التي يرتكز معظمها على نمط تصميم متحكم عرض النموذج.

حلول Ruby on Rails: لغة البرمجة Ruby هي لغة برمجة ديناميكية كائنية التوجه[27] أصبحت شائعة لكتابة تطبيقات الويب. وهي تدعم كتابة برامج كاملة ومخطوطات يمكن أن تكون جزءاً من ملفات HTML. ثمة مجتمع نشط وديناميكي لهذه اللغة عمل على إنشاء العديد من الحلول الخفيفة، ومن أفضلها Ruby on Rails[28]، وهو إطار عمل لتطوير تطبيقات الويب بناءً على نمط متحكم عرض النموذج. ولهذا الغرض، تعمل لغة Ruby على إضافة مجموعة من الوظائف الفاعلة كالأساسات والسجل الفعال والترحيل والتوجيه والبيئات ومساعدات أخرى عديدة.

أما الأساسات فهي فكرة عامة في حلول Ruby on Rails يستطيع المطوّر فيه إنشاء الإصدار الأول من التطبيق باستخدام المخطوطات. تُنشئ الأساسات نماذج وعروضاً ومتحكمات، حسبما هو مطلوب بناءً على مخطط قاعدة بيانات معرّف مسبقاً.

يدعم إطار العمل منهجية التطوير السريعة[5] عند بدء التطبيق بمجموعة أساسية من الملفات التي تمنحك العمليات الضرورية للنموذج وتتضمن العرض والتحرير والإنشاء والتحديث والحذف.

ActiveRecord هو مكتبة تتيح للمطوّر التفاعل مع قاعدة البيانات عن طريق إدارة كوائن Ruby فقط، وهذا يجعل عملية التطوير أسهل حيث يعمل المطوّر في بيئة Ruby بشكل كامل من دون استخدام لغة الاستعلام المهيكلة SQL.

الترحيل هو طريقة لإدراة مخطط قاعدة البيانات باستخدام مجموعة من الأنواع الخاصة بلغة Ruby. أما التغيرات التي تطرأ على مخطط قاعدة البيانات في أثناء صيانة البرمجية فتدعمها عملية الترحيل أوتوماتيكياً. يسهل التوجيه خرائط الطريق التي يستعلم عنها محدد المواقع URL للمتحكم المرغوب به والعملية التي تعالج الاستعلام.

يوفّر إطار العمل ثلاث بيئات عمل افتراضية: التطوير والاختبار والإنتاج. تبسط بيئات العمل المختلفة العمل على نفس الشيفرة البرمجية في مراحل مختلفة من مراحل التنفيذ بالتوازي. إضافة إلى ذلك، النشر على أجهزة خادم مختلفة بقواعد بيانات ونظم تشغيل مختلفة يحدد مرة واحدة، ويتم معالجته أتوماتيكياً بواسطة إطار العمل.

أطر تنفيذ أخرى: تدعم لغات البرمجة الشائعة من قبل أطر تنفيذ تطبيقات الويب الخاصة بها. مثلاً، J2EE هو إطار عمل تطبيقات الويب بلغة الجافا الذي تطور ليصبح متحكم عرض نموذج مع دعامات. تم تنفيذ نظام العرض الخاص بإطار العمل J2EE بوساطة صفحات خادم جافا JSP، أما المتحكمات والنماذج فهي عبارة عن Java Beans أو Java Servelts.

إطار العمل Seaside هو إطار فعّال يزيد من قوة ومرونة وبساطة التطبيق Smalltalk المستخدم لتطوير تطبيقات الويب. أما إطار العمل(*) فهو مبني على لغة البرمجة Python.

أمّا السمة المشتركة لأطر العمل هذه فهي أنها تدعم:

أ) متحكم عرض النموذج.

ب) المطابقة العلائقية الكائنية التي تطابق قواعد البيانات مع الكوائن بحيث يستطيع المبرمج البرمجة بلغة كائنية التوجه من دون عمليات قواعد بيانات صريحة.

ت) مولدات لإنشاء أجزاء البرامج الفرعية للتطبيق.

(*) هو إطار عمل تطبيقات ويب حر ومفتوح المصدر مكتوب بلغة البرمجة بايثون. طُوّر أصلاً لإدارة مواقع إخبارية تديرها «شركة العالم» (The World Company) وأصدر للعموم في تموز/ يوليو 2005 تحت رخصة بي إس دي. في حزيران/ يونيو 2008 أعلن عن إنشاء مؤسسة برنامج جانغو التي ستتولى تطوير جانغو في المستقبل. هدف جانغو الأساسي تسهيل إنشاء مواقع الويب المعقدة القائمة على قواعد البيانات (المترجم).

7 ـ 3 ـ 4 إصدار تطبيق الويب

إدارة الإصدارات هو جزء مجهد ومكلف في هندسة البرمجيات التقليدية. أما الوضع فيختلف ديناميكياً بالنسبة إلى تطبيقات الويب.

ففي تطبيقات سطح المكتب، تتطلب عمليات إضافة المزايا وإصلاح العيوب تركيب نسخ جديدة أو تطبيق إصدار فرعي صغير. أثبتت عملية الترقية هذه أنها صعبة لعدة أسباب: أولاً مشكلة التوزيع البسيطة؛ إذ يجب أن يعرف مستخدمو التطبيق أن الترقية متوفرة، ومن ثم عليهم استغراق وقت في التحميل والتركيب. ثانياً تكون عملية الترقية في الأغلب عرضة للأخطاء. وبمرور الوقت، تنزع عمليات تركيب البرمجيات لأن الاختلاف عن «التركيب النظيف»، وهذا يؤدي إلى عدد كبير من الحالات التي قد تتعامل معها عملية الترقية.

تتطلب إصدارات التطبيق ومكتبات قواعد البيانات والمصادر المختلفة ترقيات لتصبح أكثر تعقيداً مما قد يكون مطلوباً للانتقال من إصدار إلى الذي يليه على سبيل المثال. في حالة الخادم، يتم تبسيط ذلك كله بشكل كبير. يمكن إضافة المزايا وتصحيحات العيوب إلى تطبيق قيد العمل حالما تكون جاهزة، ويستفيد جميع المستخدمين من الترقيات فورياً. يتيح ذلك للمطوّرين التركيز على تحسين التطبيق بدلاً من التعامل مع الصيانة المكلفة والمعقدة ومع قضايا الترقية، وهذا يفيد مستخدمي التطبيق لأنهم يحصلون على وصول فوري لآخر إصدار من البرمجية.

وبناء على ذلك، فإن تطوير التطبيقات مفتوح أكثر لأساليب التطوير السريعة لأنه حتى الوحدات الوظيفية الصغيرة قد يتم توفيرها للمستخدمين فوراً بدلاً من توفيرها ضمن حزمة من الوظائف الأخرى التي تكون عرضة للجدولة التعسفية ما يعني عدم توفرها للاستخدام فورياً. مقارنة بتطبيقات سطح المكتب التي يكون لها دورات إصدار مدتها عدة شهور أو عدة سنوات، لا يكون الأمر مستغرباً إذا تم تحديث تطبيقات الويب عدة مرات في اليوم الواحد.

7 ـ 4 التوجهات الحالية

يمكننا تعريف قوتين على الأقل تؤثران في التوجهات الحالية في تطوير تطبيقات الويب. فمن ناحية، ثمة وظيفة جديدة تحكم تطوير أنواع تطبيقات ويب جديدة، ومن ناحية أخرى، هناك منهجيات هندسية جديدة بناء تطبيقات

ويب تحكم طريقة هيكلة التطبيقات، وبناءً على ذلك يتيح بناء مدى واسع من التطبيقات. في هذا القسم، نراجع ما يأتي:

أ) ظاهرة مشاركة المستخدم كمثال على الوظائف التي تؤثر في تطوير تطبيقات الويب.

ب) مفهوم «من سطح المكتب إلى الويب» كمفهوم يؤدي إلى إعادة هيكلة تطبيقات الويب.

7 ـ 4 ـ 1 توجه التطبيق: المشاركة

ما زال البعض يعرضون تطبيقات الويب للمعنيين على أنها «وسيلة للتوصيل المحتوى»[21]. ففي حين أن توفير محتوى هو أحد استخدامات تطبيقات الويب، إلا أن هناك جيلاً جديداً من تطبيقات الويب التي تضمّن المستخدمين في تلك التطبيقات عن طريق عرض مزيد من التفاعل والتواصل والمشاركة بين المستخدمين، وتدرج المستخدم لإيجاد قيمة في التطبيق. تبيّن الدراسات الحديثة[14] أن مواقع الإنترنت تعتبر «مواقع جيدة» حسب التوجهات في الأعوام الأخيرة، ذلك أنها توفّر للمستخدمين تفاعلاً أكبر من المحتوى. يُشار إلى ظاهرة تطويع المستخدم في تطور تطبيقات الويب على أنها «مشاركة المستخدم». يشير أوريلي (O'Reilly) إلى أن المشاركة هي إحدى أهم المتحكمات في الويب 2.0. نعرض في هذا القسم بعض أنواع تطبيقات الويب التي تشتق قوتها وفعاليتها من مشاركة المستخدمين.

أنظمة التدوين (Blog Systems): حسب الويكيبيديا، المدوّنة هي موقع إنترنت تتخذ المدخلات فيه الأسلوب الصحفي وتُعرض بترتيب زمني عكسي. إن قدرة القرّاء على إضافة تعليقات بطريقة تفاعلية تتيح للآخرين رؤيتها والتعليق عليها هي خاصية أساسية في مواقع التدوين. ظهرت المدوّنات أول ما ظهرت في نظام Blogger.com. المدوّنة هي موقع على الإنترنت يديره أحد المستخدمين، حيث يقوم بإضافة المحتوى عن طريق «الإرسال». يتم ترتيب هذه الإرساليات في فئات، ويمكن التعليق عليها من قبل مستخدمين آخرين. إن الحركة في المدوّنات كثيفة، حيث يربط المدوّنون مدوّنات أخرى في إرسالياتهم. وبذا فإن المدوّنات ذات كثافة روابط كبيرة جداً. هناك ما يزيد على 60 مليون مدوّنة على شبكة الإنترنت في وقتنا هذا.

ثمة مفهوم ذو صلة وثيقة بالتدوين والمدوّنات ألا وهو ملقمات البيانات الديناميكية. ظهر مؤخراً مخطط التوحيد البدائي لـ RSS(*) وأصبحت متداولة بين المستخدمين وهو يتطلب وجود مواقع للتصويت على الإنترنت لتحديث محتويات XML. تستخدم العديد من المدوّنات RSS لإشعار القرّاء بالتغيرات التي تجرى على المدوّنة. تعمل تطبيقات التجميع على دمج ملقمات RSS مختلفة لإنشاء مواقع بمحتويات أغنى. المدوّنات هي مثال واضح على ظاهرة غيّرت من نموذج نقل الأخبار. تعرض المدوّنات نموذجاً مختلفاً عن الصحف المطبوعة التقليدية. فهي توفر نموذجاً جديداً للمجتمع للوصول إلى معلومات محدّثة ومتجددة عمّا يجري في العالم، وإن يكن ذلك من دون عمليات التحرير الصحفي.

نظم الويكي (Wiki Systems): تشبه نظم الويكي كموقع wikipedia.org المدوّنات ظاهرياً لأنها ترتكز على مشاركة المستخدمين في إضافة المحتويات. هذه المكوّنات الأساسية هي عبارة عن صفحات كما في مواقع الإنترنت التقليدية، مقارنة بالمدوّنات التي تكون فيها المكوّنات الأساسية عبارة عن إرساليات (التي يمكن عرضها مع بعضها البعض ضمن الصفحات نفسها). تتيح مواقع الويكي للمستخدمين قراءة وتعديل محتويات الصفحات. إن الافتراض الأساسي هو أن مواقع الويكي تعرض معرفة الآراء عبر الزمن (أو على الأقل الآراء) لجميع المستخدمين. وكما هو الحال في المدونات، تعرض الويكي كثافة روابط عالية. إضافة إلى ذلك، تتوفر في الويكي روابط كثيرة بين الصفحات لأنها توفر بناءً بسيطاً لربط المستخدم بالصفحات، سواء تلك الموجودة أو التي سيتم إنشاؤها لاحقاً. كما توفر معظم مواقع الويكي المصادقة وترقيم الإصدارات لتقييد عمليات التحرير التي قد يقوم بها المستخدمون وذلك ليكونوا قادرين على استعادة عمليات التحرير القديمة.

نظم وضع العلامات التشاركية (Collaborative Tagging Systemes): نشير إلى وضع العلامات على أنها قدرة المستخدم على ربط المصطلحات (علامة) على صفحة ويب أو مصدر ويب. يمكن أن تُستخدم هذه العلامات لاحقاً للعثور على المصادر بدلاً من تسمية المصادر. هذا ويمكن أن تُستخدم العلامات

(*) Really Simple Syndication صيغة بيانات لنشر التلقيمات وهي وسيلة لتمكين البرمجيات والنظم المختلفة من استهلاك ما تنشره نظم غيرها من محتوى، ومن تطبيقاتها تمكين القراء من متابعة آخر أخبار المواقع من دون الحاجة إلى زيارة كلّ موقع على حدة (المترجم).

أيضاً لتصنيف المصادر. تتبع التطبيقات الحديثة التي تنظم المعلومات نموذجاً جديداً يدعى وضع العلامات التشاركية، حيث يعتمد مبدأه على افتراض بسيط هو أن المستخدمين يعرفون كيفية وصف مصادر الويب بمصطلحاتهم الخاصة وأنهم يغطون جوانب من المصادر بالتشارك أكثر من الخبراء أو الفهرسة الأوتوماتيكية. على سبيل المثال، يعتبر الموقع Flicker.com أحد تطبيقات Yahoo ويستخدم لتخزين الصور ومشاركتها وتنظيمها باستخدام المصطلحات. كما أن Del.icio.us هو تطبيق آخر من تطبيقات Yahoo التي تشير إلى صفحات الويب من خلال اكتشاف المستخدمين عندما يستعرضون الصفحات ويخصصون المصطلحات لصفحات الويب. ومن تطبيقات وضع العلامات التشاركية على صفحات الويب Bibsonomy.com وciteUlike.com اللذين ينظمان الإصدارات العلمية ويتشاركان بهما مع المستخدمين. أما الموقع RealTravel.com فهو نظام تدوين منظم باستخدام العلامات التشاركية.

الدراسة الميدانية التي أجريت حول بيانات العلامات التشاركية[13, 17] كشفت السلوك التشاركي للمستخدمين. أما الدراسات الاستقرائية[18] فتناقش نظام العلامات التشاركية على أنه نسخ تشاركية من ملاحظات ومشاهدات المستخدمين على أساس المصطلحات. أما عالم الهندسة الأكاديمية وهندسة الويب فهو يدرس الآن هذا النموذج في العديد من التطبيقات حتى تلك التي تعمل من خلال سطح المكتب.

الحوسبة البشرية (Human Computation): تمكن دراسة نظم الحوسبة البشرية في المرجع[30] بناءً على مبدأ أن بعض المهام المعقدة يمكن أن تتاثر ببساطة بالبشر، وهذا أمر صعب بالنسبة إلى الآلات. من الأمثلة الجيدة على ذلك CAPTCHA وهو اختصار «Completely Automated Public Turing Test to Tell Computers and Humans Apart» أي «اختبار تورنغ العام والمؤتمت للتمييز بين الحاسوب والإنسان» أو التأكيد المنظور(*).

(*) الكابتشا هو اختبار يستطيع من خلاله الحاسوب وضع أسئلته، وتصحيح إجاباتها، ولكنه لا يستطيع حل هذه الأسئلة، حيث لا يستطيع حلها سوى عقل بشري قادر على التمييز، وبالتالي تكون أي إجابة صحيحة على أي من أسئلة هذا الاختبار، هي إجابة لمستخدم إنسان وليست لبرنامج حاسوب. وتستخدم اختيارات الكابتشا في كثير من التطبيقات منها على سبيل المثال، الاستمارات الخاصة بإنشاء بريد إلكتروني في المواقع التي تقدم تلك الخدمة، وذلك لمنع التطبيقات الحاسوبية المبرمجة من إنشاء صناديق بريدية خاصة بها بشكل أوتوماتيكي متكرر وبأعداد هائلة، ثم استخدام هذه الصناديق في ما بعد لإرسال رسائل دعائية وغير مرغوب فيها لباقي المستخدمين (المترجم).

فالاسم يتحدث عن نفسه، ويمكنك مصادفتها في العديد من مواقع الإنترنت. بشكل أساسي، تكمن الفكرة في تشويه النص على صور ملونة والطلب من العميل الإعلام عمّا كان ذلك النص. إذا كان العميل إنساناً، فإنه سيتمكن من قراءته ببساطة؛ أما إذا كان العميل حاسوبياً، فإنه لن يتمكن من ذلك خلال فترة زمنية قصيرة. يميِّز هذا الاختبار بين الإنسان والآلة. ومن الاستخدامات الأخرى مقايسة الصور التي جعلت عملية وضع علامات تشاركية جماعية (يقوم غوغل بذلك في لعبة ESP)، وبشكل عام هي مثال على الحوسبة البشرية للفهرسة، استناداً إلى ملاحظات بسيطة مضمونها المصطلحات.

7 ـ 4 ـ 2 الانتقال من سطح المكتب إلى الويب

من التوجهات النامية في تطبيقات الويب الحديثة انتقال الوظائف التي كانت تنفذ دائماً في السابق كتطبيق تقليدي من خلال سطح المكتب لتعمل من خلال متصفح الإنترنت. من أمثلة ذلك تطبيقات معالجة النصوص ومعالجة الجداول والتقاويم والبريد الإلكتروني ومعالجات المعلومات الشخصية؛ فقد انتقلت هذه التطبيقات الأساسية للمستخدمين مؤخراً لتعمل على هيئة نماذج على الإنترنت. وهذا يعني أن الوظائف التي كانت تنفذ سابقاً من خلال سطح المكتب الخاص بكل مستخدم بشكل مستقل مدعومة بواسطة خادم الويب الآن وهي متاحة للمستخدمين من خلال متصفح الإنترنت. وهذا يتضمن أن التطبيق وبياناته يكون مركزياً وتُعرض كخدمة للمستخدم.

إن سهولة التشارك هي إحدى الفوائد الرئيسة لنقل أي تطبيق من سطح المكتب إلى الويب. فعن طريق نقل البيانات إلى موقع مركزي يمكن الوصول إليه من عدة أطراف، تمتاز تطبيقات الإنترنت بتوفير منصات تشاركية أسهل.

لا يمكن عرض الوثائق وتحريرها من قبل العديدين من دون الحاجة إلى إدارة التحويل بين الإصدارات المختلفة فحسب، بل إن التشارك ذا الزمن الحقيقي من خلال الإنترنت هو خطوة كبيرة في تسهيل التواصل والتفاعل الذي يتم عبر المسافات المتباعدة.

تحقق المركزية العديد من المزايا (إضافة إلى بعض المساوئ). فإدارة البيانات بطريقة أوتوماتيكية والنسخ الإحتياطية هي ميزة قياسية يمكن أن يوفرها تطبيق مستضاف.

عن طريق الاحتفاظ بالبيانات في خادم، يمكن وضع مخطط للنسخ الاحتياطي ليخدم آلاف المستخدمين في الوقت نفسه، وذلك خلافاً للحالة التي يقوم فيها كل مستخدم بإدارة نسخه الاحتياطية الخاصة عند تخزين البيانات محلياً. هذا النموذج لا يتطلب وجود ضوابط للوصولية لخادم البيانات ووجود أمن كافٍ، بل قد يثبت في نهاية الأمر أن أمر الحفاظ على الأمن خلال عدد صغير من الخوادم أسهل منه في حالة عدد كبير من حواسيب المستخدمين التي يتضمن كلٌّ منها نُسَخاً من بيانات حساسة.

إذاً، ما الذي يعيق انتقال كل التطبيقات إلى الإنترنت؟ يمكن تقسيم هذه القضايا إلى قسمين أساسيين: التفاعل واستخدام الموارد. إن الكمون الذي يحصل عن طريق تردد جميع التفاعلات ذهاباً وإياباً بين المتصفح والخادم يعني أن العديد من تطبيقات الويب لا تكون متجاوبة بقدر نظيراتها من تطبيقات سطح المكتب. إن سرعة الضوء التي توفرها موجهات الإنترنت لا يمكن التغلب عليها بسهولة من قبل المواقع التقليدية، لكن الخدمات الكبيرة المستندة إلى الويب كغوغل تطبق حلول نظم موزعة معيارية للمشكلات عن طريق تكرار خوادم التطبيقات في أرجاء العالم كافة. عندما يزيد عرض نطاق الاتصال المثالي وتقل فترة الكمون، عن طريق الانتقال إلى روابط الألياف على سبيل المثال، فإن ذلك سيحد من وقع المشكلة للجميع، لكنه لن يخدم معظم التطبيقات ذات البيانات الكثيفة. في بعض الأحوال، سيساعد الانتقال إلى نموذج نظير ـ نظير في تحسين فترة الكمون، كما ذكر في القسم 7-5-2، لكن ذلك مشكلة أساسية يتكبدها أي نوع من نظم المعالجة الموزعة. أما المصدر الأكبر للأداء الضعيف من حيث التفاعل فهو حقيقة أن منطق تطبيق الويب يمكن أن يكون منفّذاً بلغة برمجة ذات مستوى متقدم فقط كـ Javascript. على الرغم من أن ذلك أحد العوامل اليوم كما هو الحال في معظم قضايا الموارد، إلا أنه ليس من المرجح أن تستمر هذه المشكلة، بشكلها الحالي على الأقل.

طالما أن تطبيقات لغات البرمجة في تحسن مستمر وسرعة المعالجات في زيادة مطّردة، فإنه لن تكون مشكلة في المستقبل القريب للجميع، لكنها ستؤثر في التطبيقات ذات المعالجة المركزة. في الوقت الراهن، حتى أن أجهزة الهواتف الخلوية تحتوي على معالجات فاعلة جداً. الذاكرة ووسائط التخزين هي قيود إضافية على الموارد تجعل من بعض التطبيقات مقيدة بسطح المكتب؛ لكن كما هو الحال في طاقة المعالجة، زادت السعات التخزينية بمعدل جيد

بحيث يمكن الآن أن توفّر الشركات الكبرى لعملائها سعات تخزينية مجانية عالية تقدر بوحدات عديدة من جيغابايت (Gigabytes).

لا شك أن هناك مزايا ومساوئ لهذا التوجه؛ لكن حيث إن جودة الاتصال بالإنترنت وتقنيات الويب في تحسّن، فإن المساوئ الأساسية لفترة الكمون والتفاعل ستخضع غالباً لهذه التطورات التي توفرها النماذج التي تعمل من خلال الإنترنت.

7 ـ 4 ـ 3 من صفحات الويب إلى خدمات الويب

من التوجهات الأخرى التي تغيّر طريقة بناء تطبيقات الويب هو نشوء خدمات الويب. خدمة الويب هي جزء من الوظائف التي يمكن الوصول إليها من خلال الإنترنت بواجهة استخدام معيارية. يتم تشفير صيغة الرسالة بـ XML، ويتم استدعاء الخدمات بنمط اتصال إجرائي عن بعد (RPC). يتم عرض الخدمة ككائن يوفّر بَينيات للوصول إلى العمليات عن بعد. تعرض العديد من مواقع الإنترنت الآن خدمات التطبيقات الخاصة بها للمستخدمين إضافة إلى خدمات الويب، بحيث يمكن للبرامج (أي تطبيقات ويب أخرى) الوصول إلى الخدمات أوتوماتيكياً. خدمات الويب تجعل أمر بناء تطبيقات الويب ممكناً، وذلك بدمج خدمات من مواقع مختلفة عديدة. على سبيل المثال، تسهيلات البحث التي يقدمها موقع Google وقاعدة بيانات الكتب التي يقدمها موقع Amazon يمكن الوصول إليها من خلال خدمات الويب، وهكذا فإنها تعرض كمكوّنات أو خصائص أساسية للتطبيقات.

يبشّر استخدام خدمات الويب لبناء تطبيقات الويب في تحقيق العديد من أهداف هندسة البرمجيات العديدة كتكييف المكوّنات وإعادة استخدامها. في سياق تطبيقات الويب، إن تكييف المكوّنات مبدأ فعّال بلا شك، ذلك أن المكوّنات يمكن أن توفر خدمات تتراوح من المركزية العالية إلى التعميم. مثلاً، خدمة البحث التي تقدمها Google هي عامة لكن خدمة الخرائط تتيح الوصولية إلى بيانات كان جمعها مكلفاً حيث تطلب الأمر استخدام الأقمار الصناعية، وهذا غير متاح لكل مطوّري الويب.

- إن المواقع ذات كثافة البيانات العالية كـ Google وموقع تشارك الصور Flicker وغيرهما توفر لمطوّري الويب API's تتيح الوصول إلى البيانات والصور

عن بعد. وهذا يتيح للأطراف الخارجية إنشاء تطبيقات جديدة بالاستفادة من هذه البيانات وتعطي مرونة كبيرة للمستخدمين نظراً إلى سهولة الوصول إلى البيانات من أيّ مكان في العالم باستخدام متصفح واتصال بالإنترنت.

7 ـ 4 ـ 4 سطح المكتب الدلالي الاجتماعي

ظهرت قضايا تصميم الويب في تطبيقات سطح المكتب أيضاً، وذلك بظهور نظم برمجية جديدة برمجت باستخدام تقنيات الويب تعمل من خلال سطح المكتب، كوسائل لتمثيل وتواصل وتنظيم المعلومات. هذا هو الحال في سطح المكتب الدلالي الاجتماعي(8) حيث يعتبر Gnowsis تنفيذاً لها(25, 24). سطح المكتب الدلالي الاجتماعي هو امتداد لنظم التشغيل التقليدية، التي تهدف إلى إعطاء سطح المكتب أصبغة نظم الملفات الدلالية وبنية تحتية تشاركية لتطبيقات عديدة تعمل من خلال سطح المكتب للتواصل معها. مثل هذه البنية التحتية أمر مرغوب به لدعم التطبيقات الجديدة ودعم الشبكات الاجتماعية والأعمال المعرفية وإدارة واستكشاف المجتمعات ومشاركة الملفات واستكشاف المعلومات وصياغة المعرفة والتصور.

7 ـ 5 التوجهات المستقبلية

تتوسع شبكة الويب بسرعة في العديد من النواحي. ولا يُستثنى من ذلك تطوير تطبيقات الويب. من الممكن أن التقنيات الثورية ستعمل جنباً إلى جنب لتغيير النماذج الحالية. هذه الأمور لا يمكن توقعها. فبالنظر إلى التوجهات الحالية وخرائط الطريق التقنية الموضوعة لمطوّري التطبيقات الأساسية، من الممكن توقع بعض التطويرات التي يمكن إجراؤها في المستقبل القريب.

يمكننا تصنيف المجالات التي ستؤثر فيها عمليات تطوير تطبيقات الويب إلى أربعة مجالات: دعم العملاء (مثال، المتصفح) ودعم البنية التحتية (مثال، الشبكة) ومتطلبات التطبيق (مثال، الشبكات الاجتماعية) والمنهجيات التصميمية (مثال، طبقات التطبيق).

7 ـ 5 ـ 1 قضايا التصفح

لغات البرمجة المُستخدمة لبناء تطبيقات الويب تعمل على ترميز الحكمة التراكمية للمجتمع في ما يتعلق بأفضل الطرق لبناء هذه التطبيقات. بمرور

الوقت تؤثر توجهات التطبيقات والمزايا في الأجيال الجديدة من لغات البرمجة التي تسهّل عمليات تطوير المزيد من المزايا المتقدمة. أما لغات البرمجة القديمة كـ PERL وPHP أو حتى JavaScript فقد كانت تهدف أساساً إلى دعم المتصفح أو الخادم. أما لغات البرمجة الحديثة كـ Ruby فهي ذات أغراض عامة وتغطي مجال المتصفح كاملاً ومنطقية التطبيق ووظائف الخادم.

مؤخراً، أصبحت لغات البرمجة الديناميكية غير المطبوعة (كلغة Ruby) أكثر شيوعاً في مجال تطوير الويب؛ وحيث إن سرعات المُعالِج تزيد وأصبحت تنفيذات اللغة أكثر تطوراً، لا شك أن هذا التوجه مستمر. هذه اللغات حررت المطوّرين من التعامل مع تفاصيل معيّنة في الآلة ونظم التشغيل بحيث أصبحت منطقية التطبيقات تركز على التطوير. في هذا السياق، من الأهمية بمكان أن نذكر تطوير ECMAScript، وهو معيار لبرمجة صفحات الويب التفاعلية التي تتطور باستمرار لتصبح لغات برمجة ديناميكية كاملة يمكن استخدامها لتطوير البرمجيات كما في تطبيقات سطح المكتب المعقدة التقليدية. يمكن النظر إلى JavaScript على أنها بديل لـ ECMAScript.

إن لغة البرمجة أو اللغات التي تمكن الجيل القادم من تطبيقات الويب مهمة، لكن ما يوازيها أهمية هو البيئة التي تشغّل فيها هذه اللغات، ألا وهي متصفح الإنترنت. فما كان في يوم من الأيام مترجماً لملفات النصوص البسيطة التي تتضمن بعض العلامات أصبح اليوم محرك مترجم متطور منصة تطبيقات. أصبحت متصفحات الإنترنت البيئة التشغيلية الأساسية لمعظم تطبيقات الحوسبة على نحو واسع عن غير قصد، ويبدو أن ذلك هو بداية التحول في النموذج في نشر البرمجية. الانتقال إلى المرحلة الثانية سيتطلب جهوداً حثيثة.

بما إن المتصفح هو ما يوفّر جميع الإمكانيات لمطوّر تطبيقات الويب، فهو ما سيستمر في كونه محور التركيز في معظم التطوير في مجتمع الويب. من المرجح أن الوظائف طويلة المدى المضمنة في بيئة المتصفح ستحكمها التطبيقات الأكثر شيوعاً التي تعمل في الجيل القادم من تطبيقات الويب الديناميكية، لكن هناك عدداً من التطويرات التي تلوح في الأفق القريب.

لعدد من الأسباب، بما في ذلك الأمن، أعطيت متصفحات الويب وصولية محدودة للمصادر المحلية المتواجدة في جهاز الحاسوب الخاص بالمستخدمين. تكمن الفكرة في أن الوظيفة الأساسية للتطبيق تكون موجودة في الخادم، وهنا

يوفر المتصفح واجهة استخدام رسومية بسيطة (GUI) للمستخدمين للوصول إلى الوظائف المتواجدة في الخادم.

إن عدم القدرة على الاستفادة من المصادر المحلية هو عائق أمام العديد من تطبيقات الويب. الكعكات (Cookies) هي الآلية الوحيدة المستخدمة لتخزين المعلومات ضمن بيئة المتصفح المعياري، هي شكل محدود جداً من أشكال تخزين القيم الأساسية. بالبدء بالوظائف الإضافية Flash، ومن ثم تطبيقها في متصفح الإنترنت، في النسخ الأحدث من المتصفحات، فقد عززت الكعكات لتكون بمثابة نموذج تخزين يعرض في مجموعة عمل تقنية تطبيقات النصوص المتشعبة الخاصة بالويب WHATWG[9]. في هذا النموذج، يمكن أن تخزن التطبيقات بيانات مهيكلة بواجهة ذات قيمة أساسية، لكن وبخلاف الكعكات، لا ترسل البيانات إلى الخادم في كل عملية طلب لإحدى صفحات الإنترنت. في نموذج التخزين الجديد، تكون بيانات الدورة أو البيانات الشاملة متواجدة دائماً محلياً مع المتصفح، بحيث تتمكن النصوص البرمجية التي تشغل ضمن الحواسيب التابعة من الوصول إلى هذه البيانات بشكل صريح. ويتوقع أن يسهّل هذا العديد من الصعوبات البالغة في تطبيقات الويب، كما ستتيح لعدد من نماذج التطبيقات الجديدة بالظهور.

من المظاهر الأخرى لتقنيات المتصفح التي ستستمر في تحسين قدرات تطبيقات الويب هي النظم الفرعية الرسومية التي أصبحت الآن جزءاً من العديد من المتصفحات. إن علامة canava التي قدمتها شركة Apple أول مرة في متصفح Safari، هي الآن معيار من معايير مجموعة عمل تقنية تطبيقات النصوص المتشعبة الخاصة بالويب WHATWG التي ضمّنت أيضاً في Firefox[6] وOpera. إن عنصر النصوص التشعبية على مستوى الوحدة توفّر لجانب المستخدم إمكانية البرمجة بسياق الرسم ذي البُعدين المرتكز على المتجهات مشابه للحواشي التذييلية أو نموذج pdf. إن بيئة الرسومات المضمنة في المتصفح تمنح تطبيقات الويب مرونة عالية في إنشاء المحتويات التي كانت في ما سبق مقتصرة على الخادم فقط. على سبيل المثال، الرسوم البيانية والمخططات الرسومية في معالجات الجداول يمكن أن يتم إنشاؤها الآن في بيئة المتصفح لتحقيق تفاعل كبير وعرضها خارج الشبكة ما يعني إمكانية تعديلها. هذه canvas هي بداية انتقال ما كان يُعرف بمكتبات البيانات على مستوى النظام إلى بيئة المتصفح، وليس من سبب لتوقع أن يتوقف هذا التوجه إلى هذا الحد.

ثمة تحسينان واضحان إضافيان هما البيئة الرسومية ثلاثية الأبعاد ومعالج الأحداث المرتكز على canvas. ويتضمن ذلك دعم بيئة ترجمة الأبعاد الثلاثية المتسارع من حيث الأجهزة، الذي يُلْمح في كل من مواصفات مجموعة عمل تقنية تطبيقات النصوص المتشعبة الخاصة بالويب WHATWG وخريطة الطريق الخاصة بـ Firefox سيفتح الشبكة لأول مرة لتصبح بيئة ثلاثية الأبعاد حقيقية. حالياً، تنتج أداة canvas عنصراً مكافئاً لصورة على الصفحة، لكن وجود معالج الأحداث الخفيفة يتيح إنشاء محتوى أكثر ديناميكية كعناصر الواجهة المخصصة والألعاب التفاعلية.

7 ـ 5 ـ 2 البنية التحتية للشبكات

إذا استمر توجه الانتقال من سطح المكتب إلى الويب، سيكون هناك عقبة أخرى كبرى تتمثل في الحاجة إلى الاتصال الدائم بالإنترنت أو قدرة التطبيق على التعامل مع الاتصال المتقطع بالخادم. بهدف عنونة هذه القضية، تم توحيد مجموعة عمل تقنية تطبيقات النصوص المتشعبة الخاصة بالويب WHATWG في مجموعة من الأحداث التي يمكن أن تُعلم التطبيقات قيد العمل عن حالة الشبكة. وهذا يتيح لمعالجات الجداول، على سبيل المثال، تخزين معظم البيانات الحديثة محلياً بحيث يتم استعادتها حالما يتم الاتصال، وهنا لن يتم فقدان أيّ من البيانات. إضافة إلى ذلك، عن طريق إعلان التطبيق أنه خارج الاتصال، يمكن تخزين منطق البيانات والتطبيق محلياً بحيث تستمر بعض جوانب تطبيق الويب من دون الاتصال بالشبكة. في هذا النموذج، يصبح الويب آلية نشر للتطبيق وتصبح المتصفحات بيئة تنفيذية، بدلاً من أن تكون مجرد تطبيق لتصفح البيانات.

إن زيادة استخدام ملقمات RSS سار يداً بيد مع النمو السريع لمنتجات الوسائط غير المركزية على شبكة الإنترنت. فكما رأينا سابقاً، أنشأت المدونات نموذجاً جديداً لنشر الأخبار. لكن الطريقة الحالية التي تقضي بإرسال البيانات أولاً إلى موقع الإنترنت، ومن ثم يتم استقصاء رأي المستخدمين حول تحديث البيانات، لهي الطريقة المجدية الوحيدة لنشر محتوى إلى عدد كبير من الناس. لِعَنوَنة مشكلات التدرجية لطريقة نشر البيانات هذه، يجب استثمار أسلوب نشر الملفات نظير ـ نظير. ومن أشهر بروتوكولات النشر نظير ـ نظير BitTorrent الذي يمكن استخدامه لهذا الغرض. يتيح هذا البروتوكول لأي عدد من العملاء تنزيل أجزاء من المحتوى في الوقت ذاته عن طريق إنشاء شجرة مشاركة كبيرة

غير متبلورة. يعمل كلٌّ من متصفحي Opera وFirefox[7] على دمج بروتوكول التنزيل نظير ـ نظير بحيث يمكن التزيل مباشرة من خلال المتصفح بشكل متواصل. ما هذه إلا البداية لتكامل نظير ـ نظير (P2P) في المتصفح. في المتصفحات المستقبلية أو مساعدات المتصفح، من المرجح دمج أنواع أخرى عديدة من خدمات P2P. ستنتقل البيانات التي كانت تنشر في خادم ويب مركزي وحيد إلى شبكات P2P موزعة تتيح سرعة الوصول إلى البيانات، إضافة إلى قابلية عالية غير محدودة.

تتطلب تطبيقات الويب أيضاً، إضافة إلى الوصولية السريعة لبيانات الويب، كمية كبيرة من السعة التخزينية. أحد الأمثلة على ذلك الخدمة الحديثة التي تتصرف كمثال على مصدر بيانات من طرف خارجي خدمة التخزين البسيطة التي تقدمها Amazon أو S3. في هذه الخدمة، توفر أمازون شبكتها التخزينية العالمية لمطوّري الويب كأداة ذات أغراض عامة مع الاحتياجات التخزينية القابلة للقياس. يمكن تخزين البيانات واستعادتها من خدمة S3 بفضل واجهة خدمات الويب البسيطة التي يمكن الوصول إليها من خلال تطبيق خادم أو المتصفح.

7 ـ 5 ـ 3 تصميم الويب

يجب أن توفر تطبيقات الويب موثوقية وأداء ومتطلبات الجودة شأنها شأن أي منتجات برمجية أخرى. يتعامل مجال تصميم الويب مع تطوير تطبيقات الويب بطريقة ممنهجة. لكن ثمة مشكلات تتعلق بتطوير الويب تحدث عنها جينايج Ginige وتشاير Chair عام 1999[12, 4] التصفح والوصولية والموثوقية، ولا تزال هذه المشكلات واقعاً. إن استخدام أطر العمل الخاصة بالويب كتلك التي عرضنا لها في الأقسام السابقة يساعد (كنموذج لإعادة استخدام المكوّنات) على عَنوَنة معظم هذه القضايا ويؤدي إلى تحسين مطرد في تطوير تطبيقات الويب. في هذا القسم، عرضنا منهجيتين تبحثان عن طرق جديدة لتصميم تطبيقات الويب.

مواصفات الخدمات (Specification of Services): تعرض شبكة الويب حالياً طيفاً كبيراً من الخدمات التي جمعتها تطبيقات الويب لعرض المزيد من الخدمات الشاملة والمفيدة للمستخدمين. إن المنهجية التركيبية لخدمات الويب قد خدمت مجتمع خدمات الويب جيداً. لكن يتفاعل عدد كبير من الخدمات

والتطبيقات بالعديد من الطريق غير الواضحة للعيان. إن فهم التفاعلات الشاملة والعمليات وهيكليات البيانات في نظم الويب أمر صعب من وجهة نظر خارجية، وسبب ذلك غالباً، أن المطوّر لا يملك وصولية للشيفرة البرمجية، أو بسبب أن الشيفرة البرمجية كبيرة جداً بحيث يستغرق المرء وقتاً طويلاً لفهم فكرة الوظائف التي تؤديها. نحن نرى حاجة إلى مواصفات دقيقة لتطبيقات الويب وخدمات الويب كمهمة مهمة. إن تحديد نظم الويب يمكن أن يساعد كجسر بين المتطلبات والتطبيق وكمرجع لفهم النظام. لقد اختبرنا باستخدام المواصفات المنطقية المؤقتة[16] في حالة وضع علامات المشاركة. إن الحصول على مواصفة من صفحة واحدة كوسيلة تواصل مع النظام كاملاً يجعل من السهل التفاعل مع المطوّرين والمعنيين في مشاركة الفهم نفسه عن النظام.

الويب الدلالي (Sematic Web): معظم البيانات الحالية المتاحة على شبكة الويب تكون ملفات معزولة تربط في ما بينها بعض الروابط أحياناً. تُنشأ هذه الوثائق في المقام الأول لتعالج من قبل المستخدمين. أما هدف الويب الدلالي فهو إثراء هذه البيانات بحيث يمكن معالجتها أيضاً بواسطة برامج الحاسوب. الفكرة هنا هي زيادة البيانات المتاحة مع بعض المعلومات الإضافية (بيانات وصفية) لتصف ماهية البيانات وسببها. يُنظر إلى هذه البيانات الإضافية على أنها توفير «دلالات» للبيانات. من الواضح أنه إذا كانت هذه المعلومات التي يمكن الوصول إليها آلياً متوفرة، تصبح إمكانية توفير جيل جديد كامل من تطبيقات الويب متاحة. يمكن أن تتبادل التطبيقات البيانات ودلالاتها، وبذا تجنب الكثير من قضايا التوافقية.

أساسيات الويب الدلالي (Semantic Web Fundamentals): في قسم أساسيات الويب، عرّفنا محددات مواقع المصادر الموحدة URL على أنه جوهر التفاعلات في الويب كنظام، كما عرّفنا لغة الويب على أنها HTML. أما جوهر الويب الدلالي فهو معرّف المصادر الموحد URI وهو نموذج معمم لمحددات مواقع المصادر الموحدة، أما لغته فهي لغة إطار عمل وصف المصادر RDF. أما لغة الويب الدلالي فهي لغة التفاعل التي تدعم التطبيقات المهمة التي تزيد يوماً بعد يوم، والتي توفر تفاعلاً في محتواها بين التطبيقات والمستخدمين. أما رؤية الويب الدلالي فهي دعم مثل هذا التفاعل كآلية أساسية.

إن محاولات تقديم ملخصات دلالية أخرى مضمنة في وثائق HTML

كالصيغ الجزئية[15] أو لغة إطار عمل وصف المصادر RDF تبدو حلولاً مؤقتة جيدة. من ناحية أخرى، تبدو شبكيات المعلومات[1] مثالاً جيداً على طريقة حزم الويب الدلالي عن طريق توفير وصولية لنظم المعلومات المتوارثة باستخدام معايير الويب الدلالي.

رؤية الهندسة للويب الدلالي (Engineering Vision of the Semantic Web) : قمنا بوصف نمط تصميم متحكم عرض النموذج في القسم 7-3-2. حالياً، تستخدم العديد من أطر عمل تطبيقات الويب هذا النمط بنجاح. لقد استغرق تحول هذا النمط إلى معيار في تطبيقات الويب وقتاً طويلاً وتطلّب إجراء العديد من التجارب. ماذا عن حالة الويب؟ ماذا لو صممنا الويب كاملاً حسب نمط التصميم هذا؟ ما من شك أن ذلك سيحل العديد من مشكلات التصميم كالتوافقية والتناغم وسهولة الوصول، وغير ذلك. الويب الدلالي مصمم بنمط تصميم متحكم عرض النموذج. النموذج معقد تماماً لكنه متوحد في طريقة تقديمه. يمكن اعتبار النموذج على أنه لغة إطار عمل وصف المصادر RDF لمعرّف المصادر الموحد URI والمتحكمات هي تطبيقات ويب، أما ما يمثله فهو صفحات XHTML الناتجة، وذلك بخلاف الوضع الحالي لمجموعة نماذج الويب كاملة في قواعد البيانات ونظم قواعد البيانات المختلفة وصيغ الملفات وغيرها. سنكون قادرين على الوصول المباشر إلى أيّ نموذج يقع على بعد وذلك من أيّ تطبيق (بتوفر تمثيل كائنية التوجه)، بغضّ النظر عن قضايا الشبكات وتحولات البيانات والهيكلية والصيغ المختلفة. لا شك أن هذا عرض ملخص متقدم وسيتم تطبيقه بعد عدة سنوات. لكن كرؤية، يمكن أن يقود عملية إنشاء تطبيقات ذات هيكلية جيدة وجديدة.

تطبيقات الويب الدلالي (Semantic Web Applications) : لبناء تطبيقات الويب الدلالي ذات الهيكلية المحددة جيداً بصورة روتينية، من الضرورة بمكان توفّر أطر عمل لتطبيقات الويب الدلالية ذات التصميم الجيد التي توفّر بنية تحتية للويب الدلالي. أحد الأمثلة على هذه البنية التحتية خادم التطبيقات للويب الدلالي KAON[20]. حالياً، تظهر تطبيقات الويب الدلالي بالصورة البدائية كتوسع لتطبيقات موجودة حالياً كالمدونات الدلالية والويكي الدلالي[29] والمخازن الدلالية وسجلات العناوين الدلالية.

من دراسات الحالة لتطبيقات الويب الدلالية تلك الخاصة بالتراث

الثقافي[26، 23]، التي توفّر القدرة على التصفح والروابط والوصف الكامل لمعالم العالم الفنية.

7 ـ 6 الملخص والاستنتاجات

في هذا الفصل، قمنا بمراجعة التوجهات الحالية والمستقبلية في تطوير تطبيقات الويب. ألقينا نظرة على مشاركة المستخدمين كمتحكم أساسي في التطبيقات مستقبلاً. لقد اختبرنا تطوير تطبيقات الويب من وجهة نظر هندسة البرمجيات. وقمنا بتقييم تطوير البنية التحتية التي تقود إعادة هيكلة تطبيقات الويب. بعض التوجهات واضحة: تنتقل بعض التطبيقات ذات كثافة البيانات الكبيرة من سطح المكتب إلى الويب. تتكون التطبيقات من خدمات موزعة بشكل واسع؛ تدعم التطبيقات التعاون والتفاعل بين المُستخدمين على نحو متزايد. من الصعوبة بمكان التوقع بما هو آت. لا شك أن البنية التحتية لشبكة الويب ذات التصميم الجيد ضرورية وأن الويب الدلالي هو طريقة طموحة لتحقيق ذلك. على أقل تقدير، لابدّ من توفّر أنماط تصميم جديدة لتطوير تطبيقات الويب؛ حيث يجب أن توفر هذه الأنماط مستوى متقدماً من التفاعل بين المستخدمين. يجب أن تكون عملية تطوير تطبيقات الويب سريعة، كما يجب أن تستمر عمليات التطوير وأطر العمل لتحقيق هذا المتطلب.

تمّت عمليات تطوير تطبيقات الويب بسرعة لالتقاط دروس في هندسة البرمجيات، وهي الآن تقود التطبيق بأساليب السرعة لبناء نظم برمجية جديدة وموزعة بطريقة متطورة.

المراجع

1. O. Alonso, S. Banerjee, and M. Drake. «Gio: A semantic Web application using the information grid framework.» paper presented at: *Proceedings of the 15th International Conference on World Wide Web (WWW '06)*. New York: ACM Press, 2006, pp. 857-858.

2. D. Bonura, R. Culmone, and E. Merelli. «Patterns for Web applications.» paper presented at: *Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering (SEKE '02)*. New York: ACM Press, 2002, pp. 739-746.

3. S. Casteleyn [et al.]. «From adaptation engineering to aspect-oriented context-dependency.» paper presented at: *Proceedings of the 15th International Conference on World Wide Web (WWW '06)*. New York: ACM Press, 2006, pp. 897- 898.

4. S. M. Chair. «Web engineering.» *SIGWEB Newsletter*: vol. 8, no. 3, 1999, pp. 28-32.

5. A. Cockburn. *Agile Software Development*. Boston, MA: Longman Publishing Co., 2002.

6. Community. *Firefox Feature Brainstorming*, 2006.

7. Mozilla Community. *MozTorrent Plugin*, 2006.

8. S. Decker and M. R. Frank. «The networked semantic desktop.» In C. Bussler, S. Decker, D. Schwabe, O. Pastor, editors. *Proceedings of the WWW2004 Workshop on Application Design, Development and Implementation issues in the Semantic Web*, CEUR Workshop Proceedings, 2002.

9. Browser development community. *Web Hypertext Application Technology Working Group*, 2006.

10. M. Gaedke and J. Rehse. «Supporting compositional reuse in component-based web engineering.» paper presented at: *Proceedings of the 2000 ACM Symposium on Applied Computing (SAC '00)*. New York: ACM Press, 2000, pp. 927-933.

11. M. Gaedke, C. Segor, and H.-W. Gellersen. WCML: «Paving the way for reuse in objectoriented web engineering.» paper presented at: *Proceedings of the 2000 ACM Symposium on Applied Computing (SAC '00)*. New York: ACM Press, 2000, pp. 748-755.

12. A. Ginige and S. Murugesan. Guest editors' introduction: «Web engineering an introduction.» *IEEE MultiMedia*: vol. 8, no. 1, 2001, pp. 14-18.

13. S. Golder and B. A. Huberman. «The Structure of Collaborative Tagging Systems.» *Journal of Information Science*: vol. 32, no. 2, 2006, pp. 198-208.

14. M. Y. Ivory and R. Megraw. «Evolution of web site design patterns.» *ACM Transactions on Information Systems*: vol. 23, no. 4, 2005, pp. 463-497.

15. R. Khare and T. Çelik. «Microformate: A pragmatic path to the semantic web.» paper presented at: *Proceedings of the 15th International Conference on World Wide Web (WWW '06)*. New York: ACM Press, 2006, pp. 865-866.

16. L. Lamport. *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*. Boston, MA: Longman Publishing Co., 2002.

17. C. Marlow [et al.]. «Position Paper. Tagging, taxonomy, flickr, article, to-Read.» paper presented at: *Proceedings of the 17th ACM Conference on Hypertext and Hypermedia (HYPERTEXT 2006)*, 2006, pp. 31-40.

18. C. Mesnage and M. Jazayeri. «Specifying the collaborative tagging system.» paper presented at: *Proceedings of the 1st Semantic Authoring and Annotation Workshop (SAW '06)*, 2006.

19. T. N. Nguyen. «Model-based version and configuration management for a web engineering lifecycle.» paper presented at: *Proceedings of the 15th International Conference on World Wide Web (WWW '06)*. New York: ACM Press, 2006, pp. 437-446.

20. D. Oberle [et al.]. «Supporting Application Development in the Semantic Web.» *ACM Transactions on Internet Technology*: vol. 5, no. 2, 2005, pp. 328-358.

21. V. Perrone, D. Bolchini, and P. Paolini. «A stakeholders centered approach for conceptual modeling of communication-intensive applications.» paper presented at: *Proceedings of the 23rd Annual International Conference on Design of Communication (SIGDOC '05)*. New York: ACM Press, 2005, pp. 25-33.

22. T. Reenskaug. Models: views-controllers. Technical Note, Xerox PARC, 1979.

23. L. Rutledge, L. Aroyo, and N. Stash. «Determining user interests about museum collections.» paper presented at: *Proceedings of the 15th International Conference on World Wide Web (WWW '06)*. New York: ACM Press, 2006, pp. 855-856.

24. L. Sauermann, A. Bernardi, and A. Dengel. «Overview and outlook on the semantic desktop.» paper presented at: *Proceedings of the 1st Workshop on The Semantic Desktop at the ISWC 2005 Conference*, 2005.

25. L. Sauermann [et al.]. «Semantic desktop 2.0: The Gnowsis experience.» paper presented at: *Proceedings of the 5th International Semantic Web Conference (ISWC 2006)*, LCNS 4273. Berlin: Springer, 2006, pp. 887-900.

26. P. Sinclair [et al.]. «Semantic Web integration of cultural heritage sources.» paper presented at: *Proceeding of the 15th International Conference on World Wide Web (WWW '06)*. New York: ACM Press, 2006, pp. 1047-1048.

27. D. Thomas, C. Fowler, and A. Hunt. *Ruby: The Pragmatic Programmer's Guide*. 2nd ed. Raleigh, North Carolina; Dallas, Texas: The Pragmatic Programmers, 2006.

28. D. Thomas [et al.]. *Agile Web* Development with Rails: A Pragmatic Guide. Second Edition. The Pragmatic Programmers, 2006.

29. M. Völkel [et al.]. «Semantic wikipedia.» paper presented at: *Proceedings of the 15th International Conference on World Wide Web (WWW '06)*. New York: ACM Press, 2006, pp. 585-594.

30. L. von Ahn. *Human Computation*. (Ph.D. thesis). Pittsburgh PA: School of Computer Science, Carnegie Mellon University, 2005.

الجزء الثالث

تقنيات تطوّر البرمجيات

8

الترحيل إلى خدمات الويب

هاري م. سنيد (Harry M. Sneed)

8 ـ 1 القوى التي تقود عملية الترحيل

من حيث المبدأ، يقف مستخدمو تكنولوجيا المعلومات موقفاً ثابتاً من حالة الترحيل. فمن الصعوبة أن ينجحوا في الانتقال إلى بيئة تقنية جديدة قبل أن تصبح تلك البيئة بائدة، وقبل أن يواجَهوا بضرورة الانتقال مرة أخرى. ثمة قوتان تقودان عملية الترحيل:

- القوة الأولى تُغيّر تقنية المعلومات بثبات.
- القوة الأخرى تُغيّر عالم الأعمال بثبات.

هناك بالطبع علاقة معقدة بين القضيتين، ما يجعل أمر التعامل معهما كلٌ على حدة أمراً صعباً، لكن هذا الفصل ضروري لفهم ترابطهما[3].

8 ـ 1 ـ 1 تغيّر التكنولوجيا

تتغير تقنية المعلومات بمعدل مرة كل خمس سنوات. في سبعينيات وثمانينيات القرن الماضي، كان معدل التغيير مرة كل عشر سنوات. إن سبب هذا التقصير في دورة حياة التقنية ليس تزويد المُستخدمين بوظائف أكثر فحسب، بل لتلبية احتياجات سوق رأس المال أيضاً. يرزح مزوّدو تقنيات البرمجيات تحت ضغط كبير من المساهمين لزيادة مبيعاتهم ولرفع أرباح الشركات. لغاية الآن، الطريقة الوحيدة لعمل ذلك هي تقديم منتجات جديدة وإقناع المستخدمين بشرائها[16].

نشر مزوّدو البرمجيات ثقافة أن كل شيء جديد جميل، وأن كل شيء قديم بشع، مدعومين بنزوات التقنية من العالم الأكاديمي. حتى يحظى المزود باحترام أقرانه ومستخدمي تكنولوجيا المعلومات الآخرين، يجب أن يكون رائداً من رواد التكنولوجيا. أما أولئك الذين يتمسكون بالتكنولوجيا القديمة فيعامَلون بازدراء. وهذا ينطبق على الأفراد والشركات على حد سواء. فكلاهما يجري التلاعب بهما من قبل عمليات التسويق التي يقوم بها المزودون والحاجة إلى مواكبة نظرائهم. إن حقيقة أن التكنولوجيا الجديدة تحسن من أدائهم لهو موضع تساؤل. ثمة شكوك تساور نيكولاس كار Nicholas Carr وغيره من الخبراء في مجال الأعمال بهذا الشأن[10].

الحقيقة هي أن مستخدمي تكنولوجيا المعلومات يشعرون بأنهم مجبرون على الانتقال إلى تكنولوجيا جديدة حتى من دون وجود دليل على أنها ستُخفض من التكاليف وتزيد من الإنتاجية، بل إنها تدفع إلى مصير مجهول وهم يثقون ثقة عمياء أنهم على الطريق الصحيح. ليس من مستخدم لتكنولوجيا المعلومات يرغب في البقاء بعيداً عن القطع خوفاً من البقاء معزولاً أو خوفاً من أن يُعتبر غريباً. يبدو أنهم يتقبلون حقيقة أن الترحيل أمر لا مفر منه.

بزيادة تعقّد التكنولوجيا الجيدة، فإن فوائد استخدامها زادت بشكل واضح. أصبحت مزايا التكنولوجيا الجديدة التي تفوق بها مزايا التكنولوجيا القديمة أقل وضوحاً من ذي قبل، لذا يتوجب على المزودين إنفاق المزيد من الأموال في السوق لإقناع المستخدمين أن التغيير جيد لهم. أصبحت فكرة العائد على الاستثمار قضية أساسية في تقديم التكنولوجيات الجديدة، لكن غالباً ما يتم التلاعب لصالح التكنولوجيا الجديدة[40].

في بدايات الحوسبة، كانت التغيرات في الأجهزة والمعدات هي ما يقود عمليات الترحيل. كانت البرمجيات ولغات البرمجة أقل أو أكثر اعتماداً على الأجهزة التي كانت تعمل عليها. كانت تكاليف البرمجيات أقل مقارنة بتكاليف أخرى. فقد كان يمكن تشغيل برامج فورتران (Fortran) وكوبول (COBOL) في حزمة واحدة، وكان يمكن معالجة البطاقات المثقبة والأشرطة المغنطيسية ووحدات الأقراص القابلة للإزالة. أما حديثاً، فقد أصبحت التغيرات التكنولوجية أكثر من مجرد قضية برمجية؛ فلغات البرمجة والوسائط هي ما يحكم الترحيل الآن.

بدأ هذا التحول في 1990 بالانتقال إلى نظم التابع ـ الخادم. هنا، شمل التغيير كلاً من الأجهزة والبرمجيات. رغبت كل إدارة في الشركات بالحصول على حاسوب محلي خاص بها ذي محطات عمل طرفية، لكن لا يزال الحاسوب المحلي بحاجة إلى أن يكون متصلاً بحاسوب مركز مستضيف في مكان ما بقاعدة بيانات شاملة. إن الاتصال بالعديد من نقاط الاتصال في بيئة التابع ـ الخادم أصبح مشكلة كبرى ويؤدي إلى الاستعانة بالوسطاء. في هذه البيئة موزعة الأجهزة، أصبح من الضروري توزيع البرمجيات أيضاً، وهذا جعل من الضرورة تحويل نموذج البرمجة من نموذج إجرائي إلى نموذج ذي توجه كائني[11].

كانت تكاليف الانتقال إلى نظم تابع ـ خادم الأساسية عبارة عن تكاليف البرمجيات فقط. أما البرامج المركزية الحالية فكان لابدّ من تحويلها أو إعادة كتابتها. فقد كان لزاماً تقليص حجم العديد منها لتناسب السعات التخزينية للخوادم المحلية[35].

ينطوي الانتقال من النظم المركزية إلى نظم التابع ـ الخادم على العديد من التقنيات المعقدة كإلغاء واجهات الاستخدام والتحول من الهيكلة الهرمية إلى قواعد البيانات العلائقية وتقليص حجم البرامج وتحوّل الإجراءات إلى اللغات كائنية التوجه[37]. كان لابدّ من اتخاذ الكثير من التنازلات لدفع عملية الترحيل في وقت مقبول بتكاليف مقبولة. غالباً ما يتم حجب برمجيات جيدة بهدف تلبية متطلبات بيئات الأجهزة الموزعة الجديدة. بذلت جهود عظيمة لتجسيم النظم، أي تحويل البرامج الإجرائية إلى مكوّنات كائنية التوجه. كانت تلك مهمة صعبة للغاية، ولم يتم حلّها بشكل مُرضٍ البتة، على الرغم من حقيقة أن العديد من الأبحاث اختصت بها[26].

معظم المُستخدمين كانوا عبارة عن محتوى في النهاية لإعادة تطبيق واجهات الاستخدام الخاصة بهم ولتحويل بياناتهم إلى قواعد بيانات علائقية. نجح القليل من المستخدمين في إعادة تطوير تطبيقاتهم القديمة بطريقة كائنية التوجه. وسبب ذلك أنهم لم يكونوا قادرين على إعادة التقاط كيف كانت النظم القديمة تعمل بالتفصيل. أما إعادة توثيق النظم القديمة إلى مستوى العملية الأساسية ومن ثم إعادة تنفيذها بهيكلية مختلفة فكانت تكاليفها باهظة جداً. بدءاً بالانتقال إلى تكنولوجيا التابع ـ الخادم، فإن جودة البرمجيات الحالية في

تطبيقات الأعمال تحولت فعلياً كنتيجة لعدم التطابق هيكلياً[15].

لم يكد مجتمع مستخدم تكنولوجيا المعلومات ينتهي من ترحيل برمجياته إلى هيكلية التابع ـ الخادم، حتى ظهرت الموجة القادمة على هيئة الإنترنت. أبطل هذا الترحيل العديد من مظاهر ترحيل نظام التابع ـ الخادم. الآن، كان من المهم وجود جميع هذه المكوّنات ضمن جهاز واحد مشترك بحيث يتمكن جميع مستخدمو الإنترنت من الوصول إليها بسهولة. كان ذلك يعني نقل البرمجيات الموزعة إلى حاسوب مستضيف واحد فحسب. إن طبيعة البرامج بقيت كائنية التوجه، لكن كان لابدّ من أن يتم الوصول إلى العمليات مباشرة من الخارج. أدى ذلك إلى تدمير الغرض من التسلسل الهرمي للنوع. إضافة إلى ذلك، كان لابدّ من استبدال واجهات الاستخدام التي نفّذت باستخدام أدوات واجهات الاستخدام الرسومية GUI بصفحات الويب المنفذة بلغة HTML. بل أكثر من ذلك، كان لابدّ من جميع البيانات الموزعة مرة أخرى في قاعدة بيانات مشتركة عامة ذات قابلية للوصول.

كان على معظم مستخدمي تكنولوجيا المعلومات إكمال الانتقال من نظم التابع ـ الخادم إلى شبكة الإنترنت عندما أصبحت التكنولوجيا الأحدث متوفرة، تحديداً الهيكلية خدمية التوجه. الآن، أصبح من الضروري تقسيم البرمجية إلى مكوّنات خدمية مخزنة في خوادم شبكة ذات واجهة خدمة ويب معيارية. يجب أن تستبدل النظم التابعة بإجراءات عمليات الأعمال المبرمجة بلغة تنفيذ عمليات الأعمال BPEL، التي تقود العملية وتعرض البيانات وتقبلها من خلال صفحات الأنماط المتسلسلة، وتنفذ خدمات الويب. الآن، كل عملية من العمليات المحلية يجب أن تحافظ على حالتها وعلى بياناتها، بما إن خدمات الويب يجب أن تكون بلا حالة وبلا وصولية بالنسبة إلى مستخدمي قاعدة البيانات. تتطلب الهيكلية خدمية التوجه انفصالاً جذرياً عن التقنيات السابقة وإعادة تجديد تطبيقات الأعمال[5].

الهيكلية خدمية التوجه هي ثالث تغير كبير في نموذج التكنولوجيا في الخمس عشرة سنة الأخيرة، التغيرات المدفوعة بحاجة صناعة البرمجيات إلى تحقيق مكاسب جديدة. إن مستخدمي تكنولوجيا المعلومات ما هم إلا ضحايا لصناعة تكنولوجيا المعلومات سريعة التقلب، التي تسعى من خلال استخدامها إلى الحفاظ على الإيرادات بتقديم تقنيات جديدة باستمرار. إذا تابع مستخدم ما

تغيرات التقنيات، فلابدّ أن يكون في حالة ترحيل مستمرة من تقنية إلى أخرى. في الحقيقة، إن الترحيل بين التقنيات حالة دائمة في معظم الشركات الكبرى، التي أسست وحدات تنظيمية لهذا الغرض فحسب، إضافة إلى مراكز البرمجيات العديدة بما فيها المراكز الجديدة في الهند التي تزدهر بفضل هذه الترحيلات التقنية[4].

8 ـ 1 ـ 2 تغيّر الأعمال

إضافة إلى الضغط الذي تسببه تغيرات التكنولوجيا، ثمة قوى أخرى تقود تغير الأعمال، تحولت الشركات من البنى الهيكلية الكلاسيكية العميقة التي شاعت في عقد السبعينيات، إلى وحدات الأعمال الموزعة التي ظهرت في عقد التسعينيات ومنها إلى التنظيميات ذات التوجه القائم على العمليات والشبكات والقائمة على الحوسبة بشكل كامل. إن التغيرات التي تطرأ على الأعمال اليوم مقترنة بالتغيرات التي تطرأ على تكنولوجيا المعلومات. من دون الإنترنت، لن يكون من الممكن أن تُدير الشركات عملياتها ذات المراكز الموزعة في مناطق عديدة في العالم، كاستحالة تقسيم الأعمال إلى وحدات أعمال مفردة من دون ربطها مع بعضها البعض من خلال أجهزة خوادم قواعد بيانات مشتركة في هيكلية الخادم ـ التابع. في هذا الصدد، تُكمِّل التغيرات التي تُجرى على الأعمال وعلى التكنولوجيا بعضها البعض الآخر[1].

التغيرات في الأعمال تقودها الرغبة في إدارة الأعمال بكفاءة من حيث التكاليف. إن ثورة إعادة تصميم الأعمال التي قادها هامر Hammers ونشامبي Champy تهدف إلى هيكلة الأعمال تبعاً لعمليات الأعمال[18]. فبدلاً من تقسيم مسؤولية عملية معيّنة بين العديد من الوحدات المختلفة تختص كل منها باختصاصات معيّنة، كانت وحدة واحدة فقط مسؤولة عن العملية كلها منذ بدايتها إلى نهايتها. هذا موضح أفضل ما يكون في مثال التعامل مع المسافرين في رحلات الطيران. سابقاً، كان ثمة موظف واحد يعمل في الخارج للتحقق من المسافرين، وموظف آخر في الداخل لاصطحاب المسافرين إلى الطائرة. أما اليوم، فهناك موظف واحد يتحقق من المسافرين ويصطحبهم إلى الطائرة. هذا الموظف مسؤول عن عملية تحميل الطائرة بالكامل. أما العملية الموازية التي تتعلق بأمتعة المسافرين، فتكون من مسؤولية موظفين آخرين.

إن تخصيص مسؤولية عمليات الأعمال لوحدة واحدة أو شخص واحد من

شأنها أن تسهل ضمان مساءلة المسؤولين وقياس الأداء. من الواضح أن وحدة الأعمال المسؤولة عن العمليات المحلية لا ترغب في الاعتماد على دائرة تكنولوجيا المعلومات العامة لإنجاز عملها. فالمدير المسؤول يرغب في أن تكون لديه وحدة معلوماتية محلية خاصة به ويشرف عليها. إن هيكلية النظم الموزعة توفّر هذا الحل. في النهاية، يكون لكلّ وحدة أعمال جهاز خادم محلي وفريق عمل مختص بتكنولوجيا المعلومات يقدر على العمل حسب الطلب. على الرغم من أن إنتاجية وحدات الأعمال المفردة زادت، فقد زاد إجمالي تكاليف الملكية أيضاً. أما تكاليف صيانة الحلول المختلفة لكل نشاط فقد أصبحت أكبر من تكاليف المشاركة في الموارد[36].

يعِد مفهوم تكنولوجيا المعلومات خدمية التوجه بمعالجة هذه المشكلة. فهي محاولة لتوحيد مفهوم وحدات الأعمال الموزعة المستقلة، التي تعالج كلٌّ منها عملية مختلفة مع مفهوم استخدام موارد تكنولوجيا المعلومات المشتركة، التي يمكن تطبيقها في عمليات عديدة. إن الهيكلية خدمية التوجه (SOA) في حقيقتها قضية أعمال أكثر مما هي قضية تكنولوجية. يتطلب عالم الأعمال اليوم مرونة عالية جداً. يجب أن تتكيف عمليات الأعمال باستمرار لتتواءم مع تغيير متطلبات العملاء. ليس هناك المزيد من الوقت لبدء مشاريع تطوير مكثفة ومكلفة. حتى لو كانت تتبع المنهجية السريعة فإنها ستتطلب وقتاً لتسويق منتج قابل للعمل، كما إن نتيجتها لا تكون مؤكدة. تتطلب البرمجيات المفيدة وقتاً لتنضج، وهذا وقت لا تملكه معظم وحدات الأعمال المتنافسة. ثمة حاجة ملحّة لمفهوم «برمجيات عند الطلب». وهذا يعني أن الوظائف الأساسية يجب أن تكون متوفّرة قبل أن يكون هناك طلب عليها. كل ما على مستخدمي الأعمال عمله هو الاختيار من قائمة كبيرة من الخدمات البرمجية (الوصف العام والاكتشاف والتكامل UDDI) أي تلك الوظائف التي تتطلبها لعمليات الأعمال الخاصة بها ولتجهيز إجراءات التحكم لاستدعاء الخدمات الجاهزة والارتباط بها باستخدام لغة عمليات الأعمال[21].

8 ـ 2 ظهور خدمات الويب

هذا يغير دور شركات تكنولوجيا المعلومات من إجراء برمجي إلى وسيط برمجي. إن مهمة تكنولوجيا المعلومات هي التحقق من توفّر الخدمات المطلوبة عند الحاجة إليها (أي توفر المكوّنات البرمجية المكوّنة للخدمة البرمجية). وهذا

يعني إعداد مستودع للمكوّنات البرمجية التي يمكن إعادة استخدامها، الرؤية الدائمة لعالم البرمجيات التي بدأت بمكتبات لغة الفورتران في ستينيات القرن الماضي، وتطورت إلى وظائف PL/I المدمجة ومكتبات الماكرو في عقد السبعينيات، التي استمرت مع وحدات الأعمال التي يمكن إعادة استخدامها في عقد الثمانينيات ثم تراكمت في مكتبات الأنواع (Class) في التسعينيات.

لقد كان هدف صناعة البرمجيات دائماً إعادة استخدام أكثر ما يمكن من مكوّناتها المؤكدة. إن تأثر ذلك في تكاليف التطوير أقل من تكاليف الاختبار. يتطلب الأمر جهوداً هائلة لاكتشاف جميع عيوب البرمجية وإصلاحها ولشرح أن البرمجية تؤدي الوظائف التي يُفترض أن تؤديها. لأجل ذلك، يتوجب أن تمر البرمجية عبر عملية نضج طويلة. سيكون من غير المعقول أن لا ترغب في إعادة استخدام ما ثبت جدواه بالفعل. في هذا الصدد، خدمات الويب هي سبب آخر لاعادة استخدامها بشكل متكرر[24].

أما الأمر المختلف في خدمات الويب فهو أنه يمكن أن توجد فيزيائياً في أي موقع في شبكة الإنترنت العالمية حتى أن لها واجهة استخدام ذات وصولية معيارية، وأن تطبيقها مستقل عن البيئة التي تُشغّل فيها. سابقاً، المكوّنات التي يمكن إعادة استخدامها كالأنواع المتوفرة في مكتبة الأنواع المشتركة كان لابدّ أن تكون متوافقة مع الأنواع التي تستخدمها أو تتوارثها. وهذا ينطبق أيضاً على وحدات الأعمال القياسية. لم تعد تلك هي الوضع بوجود خدمات الويب؛ إذ يمكن تطبيقها بأي لغة حتى لغة التجميع البدائية، وبأي نظام تشغيل طالما أنها قادرة على خدمة واجهة الاستخدام خاصتها.

إن استخدام واجهة الاستخدام القائمة على الرسائل مع XML وبروتوكول وصولية الكائن البسيطة SOAP تجعل من تطبيق برمجيات الخدمات مستقلة عن الأجهزة التابعة. إن لغة تعريف خدمة الويب WSDL هي لغة قائمة على XML لتفسير الرسائل التي تمرر بين الخدمة والتوابع. يجب أن تكون برمجية الخدمة قادرة على قراءة وكتابة رسائل لغة تعريف خدمة الويب. يمكن أن تتواجد خدمات الويب في أي نقطة في الشبكة وهذا ممكن بواسطة آليات عَنوَنة HTTP. كل خدمة لها عنوان فريد خاص، ويمكن أن تستلم رسائل من أي نقطة في الشبكة لها قدرة على الوصول إلى ذلك العنوان[39].

ليس هناك من فروق حقيقية بين خدمات الويب والروتينيات الفرعية

القياسية التي استخدمت في عقد الستينيات، عدا الميزات الخاصة بالإنترنت التي ذكرناها مسبقاً. فهي تعالج مجموعة من الطلبات التي تستخدمها لينتج من ذلك عدد من نتائج المعالجة. فإذا كانت عديمة الحالة، فإنه لن يكون لها ذاكرة خاصة بها، وهذا يعني أن البيانات الوسيطة التي تجمعها ستفقد حالما تنتهي. وهذا يعني عبئاً إضافياً من حيث صيانة حالة المعالجة الصحيحة في الجهاز التابع. أما إذا احتفظ بحالة البيانات، فإنها تصبح أكثر تعقيداً لأنه في هذه الحالة يجب أن يتم التمييز بين التوابع والحفاظ على بياناتها منفصلة. وهذه أيضاً مشكلة متكررة الحدوث، فقد وجدت في أجهزة مراقبة المعالجة عن بعد في عقد الثمانينيات، مثال ذلك CICS وIMS-DC وقد حلت تلك المشكلة بعدة طرق مختلفة[30].

أما مشكلة خدمات الويب فهي المشكلة نفسها التي واجهتها الوحدات القياسية القابلة لإعادة الاستخدام. وهذا هو السؤال الذي يطرح نفسه على التفاصيل. كم هو عدد الوظائف التي يجب أن توفرها خدمات الويب؟ على سبيل المثال، يمكن أن تكون خدمات الويب خاصة بمعالجة الحسابات البنكية، ويتضمن ذلك جميع الوظائف كفتح حساب والإيداع وسحب المال وتحويل الأموال وحساب الفوائد وإنشاء كشف حساب والتحقق من حدود الائتمان وإغلاق الحساب. وهذا سيتطلب بالطبع واجهة استخدام عالية التعقيد. ومن ناحية أخرى، قد تكون خدمات الويب مقتصرة على حساب الفائدة فحسب. وهذا يؤدي إلى الحاجة إلى واجهة استخدام أكثر بساطة[19].

سيفضل مستخدمو الأعمال الحل الثاني ـ تفاصيل أصغر ـ لأنه يوفّر لهم مرونة عالية. ستفضل إدارة تكنولوجيا المعلومات تقديم الحل الأول لأن إدارته تكون أسهل. إن وجود العديد من الرسائل سيضع حملاً كبيراً على الشبكة، ذلك أنه يجب أن تمر هذه الرسائل ذهاباً وإياباً بين التوابع والخوادم على الويب. لم تكن تلك هي الحالة عندما استخدمت الروتينات الفرعية القياسية أو الأنواع الأساسية التي تعمل في العنوان نفسه الذي يعمل فيه مستخدموها. وهذا يولد مشكلتين بحاجة إلى الحل:

- مشكلة الحفاظ على الحالة.
- مشكلة تحديد التفاصيل.

يتوجب على الباحثين التعامل مع هاتين القضيتين والتوصل إلى حلول قابلة

للتطبيق حتى تتمكن خدمات الويب من التكييف. لا يتوجب الحفاظ على الحالة فحسب، بل يجب أن يتم ضمان تلك الحالة. أما المستوى الملائم من التفاصيل فيعتمد على السياق، ويجب أن يتم تحديده لكل وضع على حدة. ليس من حل شامل، فلن يكون من السهل العثور على حلول ملائمة لهذه المشكلات.

هناك مشكلة ثالثة تتمثل في كيفية العثور على خدمات ويب في المقام الأول. لن يتم العثور عليها بغمضة عين بطبيعة الحال، بل يجب شراؤها أو استئجارها أو استعارتها أو استعادتها أو بناؤها. وهذه هي البدائل الخمسة لتوفير خدمات الويب التي ستناقش هنا.

8 ـ 3 توفير خدمات الويب

مصادر خدمات الويب الخمس هي (كما ذكرت سابقاً):

- أن يتم شراؤها.
- أن يتم استئجارها.
- أن يتم استعارتها.
- أن يتم بناؤها.
- أن يتم استرجاعها.

8 ـ 3 ـ 1 شراء خدمات الويب

يمكن شراء خدمات الويب من مزود برمجيات. فمعظم تجار حزم برمجيات تخطيط نظم البرمجيات يعرضون مكوّنات برمجية كخدمات ويب، وهذا يتيح المجال لأي مستخدم تكنولوجيا المعلومات في شرائها. وهذا ينطبق على برمجيات الأعمال التجارية والبرمجيات العلمية والهندسية التي يمكن شراؤها جاهزة للاستخدام. من فوائد خدمات الويب الجاهزة للاستخدام ما يأتي:

- متوفرة بسهولة ويسر.
- تكون مختبرة جيداً وموثوقة نسبياً.
- تكون مدعومة من قبل البائع.

يجب أن لا يستهان بالفائدتين الثانية والثالثة. إذ يتطلب الأمر بذل جهود

كبيرة لاختبار الخدمات حتى لو كانت خدمات ويب بسيطة ذات استخدامات متباينة. كما أنه من المريح معرفة أن خدمة الويب لن تتطلب صيانة بشكل دوري، ولن تحتاج إلى تعديل، وبذا لن يقلق المستخدم بشأن التعامل مع هذه الأمور.

أما مساوىء خدمات الويب الجاهزة للاستخدام فهي كما يأتي:

- تكون غالية الثمن في الأغلب.
- يكون استخدامها مقتصراً على وظائفها.
- لا يكون لدى المستخدم إمكانية لتعديلها.
- غالباً ما تكون كبيرة الحجم ومتراصة.

أما أكبر مساؤى هذه الخدمات الجاهزة فهي أنها ليست مرنة بما فيه الكفاية. إن استخدام خدمات الويب كبيرة الحجم والمُتراصة كنظم الفوترة الجاهزة للاستخدام أو برمجيات التحقق من الائتمان هو كالبناء باستخدام الجدران الجاهزة. يتوجب على مستخدم تكنولوجيا المعلومات بناء عمليات الأعمال خاصته باستخدام خدمات الويب المشتراة. على هذا النحو، تحدد خدمات الويب كيفية بناء عمليات الأعمال المرتبطة بها. وهنا فإن المُستخدم لا يشتري البرمجية فحسب، بل عمليات الأعمال كلها أيضاً.

بالنسبة إلى بعض مستخدمي تكنولوجيا المعلومات، قد يكون ذلك نعمة. فهم لن ينفقوا الوقت والجهد في تصميم عمليات أعمالهم، لكنهم سيخسرون أهم ميزة من مزايا خدمات الويب الرئيسة ألا وهي المرونة. كما قد يشترون عملية الأعمال كاملة، كما كان الحال في السابق. لا يوجد أي معنى في الانتقال إلى الهيكيلية خدمية التوجه. أما بالنسبة إلى مستخدمي تكنولوجيا المعلومات المتحمسين لتخصيص عمليات الأعمال، فهذا تقييد لا يُطاق. ما الذي سيدخل إطار المنافسة إذا كان كل منافس يستخدم عملية الأعمال نفسها؟

8 ـ 3 ـ 2 استئجار خدمات الويب

طريقة بديلة عن شراء خدمات الويب هي استئجارها. العديد من بائعي البرمجيات الجاهزة كنظم SAP وOracle يعملون الآن على تنفيذ خطط لتصبح خدماتهم متاحة على أساس التأجير. فبدلاً من الاضطرار إلى شراء الحزم البرمجية وتنصيبها في أجهزتهم الخاصة، يكون لدى مستخدم تكنولوجيا

المعلومات خيار استخدام الوظائف التي يتطلبها فقط عندما يتطلبها (أي على الطلب). ومن ثم يدفع مقابل الاستخدام الفعلي فقط. لمشروع الأعمال هذا عدة فوائد تفوق أسلوب الشراء:

- المُستخدم ليس ملزماً بتنصيب خدمات الويب في موقعه ولا تحديثها.
- يعمل المستخدم دائماً بآخر إصدار محدّث.
- يدفع المستخدم مقابل ما يستخدمه فعلياً فقط.

إن امتلاك البرمجية لا يكون ذا منفعة للمستخدم دائماً؛ إذ يجب أن يتحمل المستخدم مع تكاليف الملكية الإجمالية. كما يجب عليه أن ينصب البرمجية ويختبرها في بيئة عمله إضافة إلى تنصيب واختبار الإصدارات الجديدة منها. إن صيانة البرمجية في بيئة المستخدم ذات تكلفة عالية. أما الفائدة فهي أن المستخدم يستطيع تعديل البرمجية لتتواءم مع متطلباته الخاصة. لا يكون المستخدم ملزماً بتعديل عملية الأعمال خاصته لتلائم البرمجية المعيارية، وهي الوضع عندما يقوم المستخدم بالاستئجار فقط.

ينطبق الأمر ذاته على خدمات الويب. فإذا كان المستخدم يشتري هذه الخدمات، فإنه يمكن أن يكيّفها. أما إذا كان يستأجرها، فعليه أن يستخدمها كما هي. إذا كانت تفاصيل خدمات صغيرة بما فيه الكفاية، لن تكون هناك مشكلة كبيرة بالنسبة إلى المستخدم؛ إذ يمكنه بناء هذه الخدمات في عمليات الأعمال خاصته كبناء حجارة الحصى في الجدار الإسمنتي، لكن إذا كانت هذه الخدمات مبنية كألواح من الإسمنت الجاهز، فيجب على المستخدم تكييف خطط البناء خاصته لتناسبها من ناحية أخرى، للمستخدم الحرية في تنصيب الإصدارات الجديدة بشكل دائم واختبارها[25].

8 ـ 3 ـ 3 استعارة خدمات الويب

إن أخذ خدمات الويب من مصدر مفتوح مكافئ لاستعارتها. إن مناصري التطبيقات مفتوحة المصدر هم أولئك الذين يفضلون أن يَدَعوا الآخرين ينفذون الأعمال لهم ومن ثم يأخذونها بتغيير أو من دون تغيير. فمن ناحية، هم لا يرغبون أن يدفعوا مقابل هذا العمل، ومن ناحية أخرى، هم لا يرغبون في تطويرها. تعتبر خدمات الويب مفتوحة المصدر ملكية عامة يمكن لأي كان استخدامها.

هناك أمران هنا، الأول أدبي والآخر قانوني. الأمر الأدبي هو أن البرمجية هي ملكية فكرية بما في ذلك خدمات الويب. فقد قام أحدهم بالتضحية بوقته الثمين لإيجاد حل لمشكلة معضلة. فإذا كان هذا الحل ذا قيمة بالنسبة إلى شخص آخر، فإنه يجب أن يكون مستعداً لدفع ثمنه. وبخلاف ذلك، فإنه يخرق مبادئ اقتصاد السوق الحرة. لذا وفي هذا الصدد، إن استخدام خدمات الويب ذات المصدر المفتوح موضع شك ولا تنسجم مع المجتمع الذي نعيش فيه.

أما الأمر القانوني، فهو المسؤولية. من الذي يكون مسؤولاً عمّا تنفذه الخدمة المستعارة؟ طبعاً، لا يمكن أن يكون مؤلفو الخدمة هم المسؤولون، وذلك لأنهم غافلون عن مكان وكيفية استخدام ملكيتهم الفكرية. وبناءً على ذلك، يكون المستخدم الوحيد للملكية الفكرية هو مستخدم البضاعة المستعارة. عند أخذ خدمات الويب من مصادر مفتوحة، يكون للمستخدم حرية تكييف المصدر مع احتياجاته الخاصة، لكن يجب أن يفترض أيضاً أنه مسؤول عن تصحيحه وجودته، أي أن يقوم باختباره تماماً لجميع الاستخدامات الممكنة. معظم الأشخاص لا يدركون أن اختبار البرمجية مكافئ من حيث الوقت والتكلفة لتطويرها. وإذا كان المصدر غير مألوفٍ للأشخاص الذي يجب أن يكيّفوه ويصحّحوه، فقد يكون أغلى مما لو قاموا بتطوير تلك البرمجية بأنفسهم. بهذه الطريقة، ستكون الشيفرة البرمجية على الأقل مألوفة أكثر لهم. أثبت العديد من الدراسات أن صيانة البرمجيات هو التأثير الذي يقضون في محاولة فهمه[41].

إن فهم الشيفرة البرمجية هو الحاجز الأكبر الذي يعترض استخدام الشيفرة البرمجية للمصادر المفتوحة. لذا فإن المستخدمين يجب أن يفكروا مرتين قبل أن يبدأوا في استعارة خدمات الويب من المصادر المفتوحة. إذ قد تتحول إلى حصان طروادة.

8 ـ 3 ـ 4 بناء خدمات الويب

يمكن أن يتم تطوير خدمات الويب في مؤسسة المستخدم نفسها شأنها شأن حزم البرمجيات الأخرى، أو من قبل متعهد خارجي يتعاقد مع المؤسسة. أما الفرق في ما يخص التطبيقات البرمجية التقليدية فهو أن خدمات الويب يجب أن تكون أصغر، وأن يكون بناؤها أسهل إذا ما عرّفت جيداً. الفرق الآخر هو أن خدمات الويب هي ملكية مشتركة في المؤسسة التي توفرها؛ أي إنها

تنتمي لجميع إدارات تلك المؤسسة. هذا كسر جدي للتقاليد السابقة عن كيفية تمويل برمجية في مؤسسة ما.

في الماضي، كانت دائرة تكنولوجيا المعلومات تعتبر مأوىً برمجياً داخلياً مهمته توفير خدمات للإدارات الأخرى. فإذا أرادت دائرة التسويق نظام علاقات مع العملاء، فإنها تفوّض دائرة تكنولوجيا المعلومات بتطوير النظام أو شرائه لها. أما إذا رغبت دائرة التزويد والإمداد بنظام جديد لإدخال الطلبيات، فإنها تفوض دائرة تكنولوجيا المعلومات ببنائه لها. إن إدارات المستخدمين هي التي لها سيطرة على الميزانية وهي الوحيدة المهيأة للدفع مقابل شيء ذي منفعة مباشرة لها.

في حالة خدمات الويب، ليس واضحاً لأي الإدارات تنتمي. فأي شخص في الشبكة التنظيمية يمكنه الوصول إليها. إذاً من هو المسؤول عن الدفع مقابل الحصول عليها؟ الحقيقة هي أن خدمات الويب الجيدة تتطلب جهوداً كبيرة للتخطيط لها وتطويرها واختبارها. يجب أن تكون هذه الخدمات أكثر ثباتاً وموثوقية من النظم محدودة الاستخدام القديمة، كما يجب أن تكون قابلة لإعادة الاستخدام لأغراض مختلفة ضمن سياقات مختلفة. فهي كنظم المستخدمين الأخرى، تكلف خدمات الويب أكثر بثلاث مرات من النظام المصممة للمستخدم الواحد العادية. إن إدارات المستخدمين غير مهيّأة جيداً لتمويل المشاريع غير المخصصة لمتطلباتهم، وهذا يعد بمنفعة طويلة الأمد فقط. إذا كانت ثمة مشكلة يجب حلّها، فهم يرغبون في حلّها مباشرة وفورياً، أي أنها يجب أن تعدّل لتلبّي متطلباتهم المخصصة، ولا شيء غير ذلك.

إن تطوير خدمات الويب هو استثمار طويل الأمد[6]. فخدمات الويب ستستغرق نحو عامين قبل أن تصبح ذات جودة كافية لتصبح جاهزة لاستخدامها في عمليات الأعمال. من المشكوك فيه إذا كان المستخدمون على استعداد للانتظار طويلاً. أما منافع استخدام خدمات الويب ذات التطوير الذاتي فستظهر خلال بضع سنوات قادمة. في الوقت الحالي، يجب أن تتحمل دائرة تكنولوجيا المعلومات استمرارية النظم الحالية. وهذا يضع عبئاً مضاعفاً على عاتق المؤسسة. لهذا السبب ولأسباب أخرى، فإن تطوير خدمات الويب الخاصة قد لا يكون بديلاً جاذباً. أما الحاجز الأكبر الذي يعترض التطوير فهو الوقت اللازم، والأمر الآخر هو التمويل.

8 ـ 3 ـ 5 استعادة خدمات الويب

أما البديل الخامس والأخير فهو استعادة خدمات الويب من تطبيقات البرمجيات الموجودة. صحيح أن البرمجيات الحالية قد تكون قديمة ويصعب إدارتها، إلا أنها تعمل. ليس فقط تعمل، بل إنه يمكن تكيّفها لتلائم الإدارات المحلية. فهي تلائم البيانات وبيئة عمل المؤسسة. لذا، لماذا لا يُعاد استخدامها؟ يجب أن لا يكون الهدف استخدام التطبيقات الموجودة كلها، وذلك أنها لن تكون ملائمة لعمليات الأعمال الجديدة، ولكنها ستلائم بعض أجزاء منها. قد تكون هذه الأجزاء عبارة عن عمليات أو إجراءات أو وحدات أو مكوّنات. الأمر المهم هو أن تكون قابلة للتنفيذ بشكل مستقل. لذا يجب أن يتم طيُّها. إن تكنولوجيا الطي هي المفتاح لإعادة استخدام البرمجية الموجودة. إن اللغة التي كتبت بها البرمجية الموجودة ليست بالأمر المهم جداً، طالما أنها قابلة للتنفيذ في بيئة الخادم[2].

بما إن طلبات الحصول على خدمات الويب يمكن أن يتم إعادة توجيهها، من الجائز وجود خوادم استضافة مختلفة لأنواع لغات البرمجة المختلفة على نحو جيد. بذا، قد يكون هناك خادم للغة Cobol وآخر للغة PL/I وآخر للغة C/C++. الأمر المهم هو أن المكوّنات المستخلصة مجهزة بواجهة الاستخدام WSDL المعيارية التي تحول البيانات في الطلبات إلى بيانات محلية في البرنامج الذي يقوم بدوره بتحويل مخرجات البرنامج إلى بيانات في الطلبات. إن إنشاء مثل واجهات الاستخدام هذه يمكن تنفيذه أوتوماتيكياً بحيث لا يتطلب ذلك جهوداً إضافية أكثر من الجهد اللازم لاختبار الواجهة[32]. أما الخدمة ذاتها فقد تم اختبارها خلال سنوات الاستخدام الإنتاجية. أما أهم فائدة لهذه الطريقة فهو التكلفة المنخفضة والزمن القصير اللازمان لتصبح الخدمات جاهزة للاستخدام.

تتمثل مساوئ هذه الطريقة في النقاط الآتية:

- البرمجية تكون قديمة أو لا تكون مفهومة بسهولة.
- تحويل البيانات من الصيغة الخارجية إلى الصيغة الداخلية يقلل من الأداء.
- قد لا يكون هناك مبرمجون ممّن يألفون اللغات القديمة.

هناك مشكلة إضافية هي الحفاظ على حالة البيانات من عملية إلى أخرى

إذا كانت البرمجية غير مصممة أصلاً ليتم إعادة التشارك بها. إن إعادة التشارك يجب أن يتم تحديثها في النظام. هنا يتبادر السؤال عن كيفية الحفاظ على حالات العديد من الحالات التي يأتي بها المستخدمون، لكن تلك ليست مشكلة خاصة بطريقة استعادة خدمات الويب، بل هي مشكلة تنطبق على جميع الطرق.

8 ـ 4 التنقيب عن خدمات الويب

التركيز الأساسي لهذه المساهمة هي كيفية استعادة خدمات الويب من التطبيقات الموجودة، ويشار إلى ذلك بالتنقيب عن خدمات الويب. تم تطبيق الكثير من وظائف الأعمال فعلياً في مؤسسة مسبقاً. المشكلة هي أنه لا يمكن الوصول إليها بسهولة. الوظائف تكون مدفونة في نظم البرمجيات القديمة[9].

وحتى تكون هذه الوظائف متوافرة ليعاد استخدامها كخدمات ويب، يجب أن يتم استخراجها من السياق الذي نُفّذت فيه، ومن ثم تكييفها لتلائم المتطلبات التقنية للهيكلية خدمية التوجه. وهذا يتضمن أربعة أنشطة مختلفة:

- الاكتشاف.
- التقييم.
- الاستخراج.
- التكييف.

8 ـ 4 ـ 1 اكتشاف خدمات الويب المحتملة

من حيث المبدأ، كل وظيفة من وظائف البرنامج تنفّذها الشيفرة البرمجية القديمة هو خدمة ويب محتملة. هنا يجب أن نلاحظ أن جزءاً كبيراً من الشيفرة البرمجية القديمة مكرسة لتنفيذ بعض الوظائف التقنية أو لتخدم بعض مواقع تخزين البيانات القديمة أو تكنولوجيا تواصل البيانات. أظهرت البيانات أن ذلك يقدّر بحوالى ثلثي الشيفرة الإجمالية. وهذا يعني أن ثلث الشيفرة البرمجية متاح لتحقيق أهداف التطبيق. إنها تلك الشيفرة البرمجية التي تهمنا في عملية الاستعادة. المشكلة هي أن الشيفرة البرمجية المركزة للتطبيق متشابكة كثيراً مع الشيفرة التقنية. ففي وحدة شيفرة برمجية واحدة، أي عملية أو وحدة أو إجراء، قد يكون هناك أجزاء خاصة بإعداد قناع عرض البيانات أو أجزاء خاصة بحساب

ضريبة القيمة المضافة. وبالبحث في الشيفرة البرمجية، يجب أن تكون أداة التحليل قادرة على إدراك هذه الأجزاء التي توفّر قيمة للعمل(33).

من ناحية أخرى، ليست جميع وظائف الأعمال صحيحة، إذ يصبح العديد منها قديم خلال الوقت. لذا، لا يجب أن تكون الشيفرة البرمجية لوظائف البرنامج مُعرَّفة فحسب، بل يجب أيضاً أن يتم التحقق من كونها ذات قيمة حالياً. وهذا يؤدي إلى ظهور قضيتي بحث هما:

1. كيفية تحديد الشيفرة البرمجية التي تؤدي وظيفة البرنامج.

2. كيفية تحديد ما إذا كانت الوظيفة لا تزال ذات قيمة حالياً للمستخدم.

ستتطلب هاتان القضيتان بعض أشكال صنع القرار القائم على القواعد. الأمر المهم هو أن المستخدم قادر على ضبط القواعد لتلائم احتياجاته، أي إن أدوات التحليل يجب أن تكون ذات قابلية عالية للتخصيص. كما يجب أن تكون سريعة جداً طالما أنها ستقوم بمسح عدة ملايين من أسطر الشيفرة البرمجية لتحديد الأجزاء التي يُحتمل أن تكون خدمة ويب. لذا فالأمر المطلوب هنا هو آلة بحث معقدة لمعالجة الشيفرة البرمجية المصدرية. من المحتمل أن لا يكون ممكناً اختيار خدمات الويب المحتملة من دون مساعدة بشرية. لذا سيكون هناك أيضاً بعض التفاعل من قبل المستخدمين مع الأداة لإعطاء المستخدم فرصة التدخل في البحث واتخاذ القرارات التي لا يمكن للأداة اتخاذها. مثل أدوات التنقيب التفاعلية هذه هي موضوع بحث كبير.

كان المفتاح الأساسي لاكتشاف خدمات الويب المحتملة في الشيفرة البرمجية الموجودة قد وصف من المؤلف نفسه في ورقة عمل سابقة قدمت حول استعادة قواعد العمل(34). الهدف هو تحديد أسماء حصيلة البيانات الأساسية ولتتبع كيفية إنتاجها. يمكن تحقيق ذلك من خلال تحليل تدفق البيانات المعكوس. قد يمر تتبع تدفق البيانات من خلال عدة عمليات أو إجراءات في أنواع أو وحدات مختلفة. من الضروري تحديدها جميعاً. من الأمثلة على ذلك معدل الائتمان في النظم المصرفية. الحصيلة الأساسية هي معدل الائتمان لكن قد يكون العديد من الوحدات أو الأنواع مشتركاً في إنتاج تلك الحصيلة. بناء على ذلك، يجب أن يتم جمعها كلها لإنتاج خدمة ويب واحدة ـ أي حساب معدل الائتمان. هذه المشكلة ترتبط بمشكلة تأثير التحليل في صيانة البرمجية.

8 ـ 4 ـ 2 تقييم خدمات الويب المحتملة

من قضايا البحث الأخرى في ما يتعلق باكتشاف خدمات الويب المحتملة قضية تقييم مقاطع الشيفرة البرمجية المنتقاة كخدمات ويب محتملة. هنا، يجب أن يحدد مالك الشيفرة البرمجية ما إذا كان استخراج مقاطع الشيفرة البرمجية من النظم التي تتواجد فيها أمر يستحق العناء. وهذا سؤال يطرح حول إمكانية إعادة الاستخدام.

يجب أن يتم تطوير المقاييس التي ستشير إلى ما إذا كان بالإمكان إعادة استخدام الشيفرة البرمجية أم لا. لقد تعامل المؤلف مع هذه المشكلة من قبل وأصدر ورقة عمل حولها[28]. المقياس الأساسي هو إعادة الاستخدام. تكون قطعة من البرمجية قابلة لإعادة الاستخدام إذا كانت منفصلة بسهولة عن الشيفرة البرمجية المحيطة بها. هناك بعض الروابط الوظيفية (أي استدعاءات الوظائف في الشيفرة البرمجية المحيطة)، وهذه تتشارك في بعض البيانات مع إجراءات أخرى. وهكذا فإنه يجب عد الاستدعاءات الغريبة والفروع الأخرى خارج كتلة الشيفرة البرمجية موضع التساؤل، إضافة إلى البيانات غير المحلية ـ أي البيانات المصرح عنها خارج كتلة الشيفرة. يجب أن تكون هذه مرتبطة بحجم كتلة الشيفرة البرمجية أو الكتل مقدرة بعدد التعابير البرمجية.

قد تكون تلك نقطة بداية جيدة، لكنها ليست كافية. هناك أسئلة أخرى عن جودة الشيفرة البرمجية وقيمة الأعمال. السؤال هنا إذا ما كانت الشيفرة البرمجية جيدة بما فيه الكفاية وقيِّمة لترحيلها إلى الهيكلية الجديدة. يمكن استخدام مقاييس الجودة المتوافرة لتحقيق هذه الغاية. يجب أن تُقاس قيمة العمل من حيث مدى أهمية النتائج التي تنتج من جزء محدد من الشيفرة البرمجية. هناك أبحاث قليلة في هذا المجال. أخيراً، يجب أن يتم حساب كلفة استخراج الشيفرة البرمجية بالنسبة إلى الفوائد التي نجنيها من إعادة الاستخدام. يحتاج الأمر إلى المقاييس لقياس قابلية صيانة وقابلية اختبار وتوافقية وقابلية إعادة استخدام الشيفرة البرمجية، إضافة إلى قيمة الوظائف التي تؤديها تلك الشيفرة البرمجية.

إن تقييم خدمات الويب المحتملة ليست مسألة تافهة وتتطلب مقاييس مثبتة لإتقانها. هذا هو المكان الذي تستدعى فيه المقاييس لاستخدامها لتوجيه القرار إذا ما سيتم إعادة استخدام الشيفرة البرمجية أم لا.

8 ـ 4 ـ 3 استخراج الشيفرة البرمجية لخدمة الويب

حالما يتم تحديد أن مقطع الشيفرة البرمجية هو خدمة ويب محتملة، تكون الخطوة التالية هي استخراجها من النظام المضمنة فيها. قد يكون ذلك مهمة معقدة جداً، خصوصاً عندما لا تكون الشيفرة البرمجية عبارة عن وحدة يمكن تنفيذها بشكل منفصل. يمكن أن تشارك الوحدات الإجرائية بيانات مع وحدات أخرى في موقع التخزين الأساسي. وقد يتم استدعاء وحدات أخرى أيضاً. يجب أن تُغطى جميع هذه التبعيات حتى يمكن استخراج الشيفرة البرمجية الإجرائية من بيئاتها. إن استخراج الشيفرة البرمجية كائنية التوجه أسهل عموماً من استخراج الشيفرة البرمجية الإجرائية، لكن هناك مشكلات أيضاً هنا. فقد يتوارث نوع معيّن أنواعاً أخرى ذات مستوى أعلى، حيث لا يرغب أحدنا في استخراجها. كما قد يستدعي النوع عمليات تابعة لأنواع غريبة، وقد ينتج منها نتائج مهمة. يجب الحد من هذه التبعيات إما عن طريق تسوية النوع أو إدراج العملية. لكن ليست هذه بالحلول البسيطة. فالأبحاث مطلوبة لتحديد أفضل الوسائل لاستخراج الأنواع والشيفرة الإجرائية.

مشكلة صعبة تعوق استخراج الشيفرة البرمجية من النظم القديمة وهي استخراج الميزات[17]. في البرمجيات كائنية التوجه تحديداً، تكون الميزة مبعثرة في العديد من الأنواع المحتواة في المكوّنات العديدة. الميزة هي سلسلة من العمليات الموزعة التي تتحفز بواسطة حدث، وينتج منها مخرجات محددة مسبقاً كالرد على استفسار أو نتيجة محسوبة، على سبيل المثال سعر مادة معيّنة أو معدل الائتمان. للحصول على هذه النتيجة، يجب أن يتم تنفيذ عمليات مختلفة في أنواع مختلفة بترتيب معيّن معطى. ستتوافق خدمة ويب مقترحة مع إحدى الميزات على الأرجح. إن استخراج الميزات من المكوّنات يشكل تحدياً صعباً للباحثين في مجال البرمجيات. إن إمكانية استخراج تلك العمليات التي تجتازها الميزة فحسب أمر مشكوك فيه، ذلك أن هذه العمليات تستخدم خصائص النوع التي قد تؤثر في عمليات أخرى. من ناحية أخرى، سينتج من استخراج جميع الأنواع خدمات ويب كبيرة جداً تتضمن كمية كبيرة من الشيفرة البرمجية غير المرتبطة بمهمة خدمة الويب التي في متناول اليد. إن حل هذه المشكلة ـ إذا كانت قابلة للحل ـ سيتطلب جهوداً كبيرة من الباحثين.

8 ـ 4 ـ 4 تكييف شيفرة خدمات الويب

قضية البحث الأخيرة هي تكييف وحدات الشيفرة البرمجية المستخرجة لإعادة استخدامها كخدمات ويب. وهذا يستلزم تزويد هذه الوحدات بواجهة لغة وصف خدمات الويب WSDL. وَسِيطاتُ الإدخال التي تستلمها وحدات الشيفرة البرمجية مسبقاً من قائمة عوامل، وقناع واجهة الاستخدام، وملف الإدخال وبعض وسائل إدخال البيانات الأخرى يجب أن يعاد تخصيصها كوسيطات ضمن طلب لغة وصف خدمات الويب. وهذا يعني تحويلها من XML ونقلها من رسالة بروتوكول وصولية الكائن البسيطة SOAP المستلمة إلى موقع تخزين داخلي في خدمة الويب. نتائج المخرجات التي أعيدت من قبل على هيكلية أقنعة إخراج وقيم إرجاع وملفات مخرجات وتقارير ووسائل إخراج بيانات أخرى يجب أن يعاد تخصيصها كنتائج لاستجابة لغة وصف خدمات الويب. وهذا يعني نقلها من موقع التخزين الداخلي لخدمة الويب إلى رسالة بروتوكول وصولية الكائن البسيطة الخارجة وتحويلها إلى صيغة رموز XML[7].

هذا ويمكن أتمتة جميع هذه الخطوات. في الواقع، إن تكييف الشيفرة البرمجية هي النشاط الذي يفسح مجالاً أفضل للأتمتة. مع ذلك، لا تزال هناك حاجة إلى إيجاد أفضل الطرق لعمل ذلك. يتطلب الأمر إجراء أبحاث عن تطوير أداة للتكييف. يصف ما تبقى من هذا الفصل منهجية المؤلف لأتمتة تكييف خدمات الويب المحتملة.

8 ـ 5 تطبيق تقنيات التجهيز

ليس من طريقة فُضلى شاملة لتجهيز البرامج التطبيقية. يعتمد خيار تقنية التجهيز على نوع البرنامج. في مجال معالجة بيانات الأعمال، هناك ثلاثة أنواع برامج رئيسة:

- برامج المعاملات من خلال الإنترنت.
- برامج معالجة الإصدارات الصغيرة.
- برامج فرعية عامة[35].

لا يهم اللغة المستخدمة ما إذا كانت لغة التجميع أو PL/I أو كوبول، فجميع البرامج ذات النوع الواحد تشترك في بعض الميزات.

تُحكم برامج المعاملات التي تجرى من خلال الإنترنت بواسطة نظام متابعة معالجة عن بعد كـ IMS/DC وCICS وTuxedo. يعتمد بناء هذه النظم على نوع متابعة المعاملة. في بعض الحالات، يمتلك البرنامج التطبيقي زمام التحكم، ويستدعي نظام متابعة المعاملة لتنفيذ الخدمات له. في حالات أخرى (مثلاً CICS)، يكون التطبيق عبارة عن برنامج فرعي لنظام متابعة المعاملة وينفذ وظائفها عند استدعائه من قبل نظام المتابعة. فهو محكوم بالأحداث. يتم إدارة جميع بيانات التواصل من خلال متابعة معالجة عن بعد. البرنامج التطبيقي مزود بمؤشرات لعنْوَنَتها وتحفظ في مجال تواصل مشترك.

في الحالتين، لا تكون برامج الإنترنت مجرد برامج كوبول أو PL/I. فهي خصائص لغة كاملة مفروضة من قبل متابعة المعالجة عن بعد على هيئة تعليمات إضافية أو ماكرو أو بيانات استدعاءات خاصة. تفرض نظم CICS ماكرو خاص بها هو EXEC CICS على البرنامج المستضيف بحيث تصبح مضمنة في كتلة الشيفرة البرمجية. أما نظم IMS فتفرض هيكليات بيانات خاصة وعوامل متغيرة على البرنامج المستضيف. يجب أن يدرك محلل (parser) برامج الإنترنت هذه ويحلّل بناءات اللغة الخاصة، كما لو كان يتعامل مع بيانات لغة البرامج المستضيفة.

عند تجهيز برنامج إنترنت، من الضرورة بمكان إيقاف تشغيل جميع العمليات التي تتواصل مع البيئة، بينما يجب الإبقاء على العمليات الأخرى التي تحدد مواقع التخزين أو تعالج الرسائل التي تفيد بوجود أخطاء. المتطلب هنا هو استبدال الأقنعة بالحقول المصححة وضبط خصائصها بوثيقة من نوع XML، والإبقاء على منطق البرنامج كما كان. كما هو موضح لاحقاً، يمكن تحويل برامج الإنترنت أيضاً إلى برامج فرعية ذات واجهة ربط. مع ذلك، يستلزم ذلك تغيير هيكلية البرنامج. في كلتا الحالتين، تتضمن مهمة التجهيز توفير بيانات لبرنامج الإنترنت، ويمكن إنجاز ذلك أفضل ما يكون من خلال رد انعدام حالة البرنامج[30].

تُحكم برامج الإصدارات الصغيرة بواسطة ملفات الإدخال التي تعالجها. وقد تكون هذه الملفات عبارة عن ملفات معاملات أو ملفات عوامل تغيير. بصورة طبيعية، سيقرأ برنامج الإصدار الصغير تسلسلياً في ملف إدخال أو أكثر أو في قائمة انتظار الرسائل، معالجاً أحد السجلات أو أكثر أو الرسائل في الوقت الواحد إلى أن يصل إلى النهاية. وقد تكون النتيجة عبارة عن تحديث

قاعدة البيانات أو تدفق من رسائل المخرجات أو ملف مخرجات أو تقرير. تحدد حالة مجال الإدخال ما يجب عمله.

الهيكلية الرئيسة لبرنامج الإصدار الصغير هي عبارة عن حلقات متداخلة، كل واحدة تمثل مستوىً دلالياً واحداً من البيانات كما اعترف بذلك مايكل. أ. جاكسون (Michael. A. Jackson) منذ فترة طويلة[20]. يتم إنهاء الحلقات بواسطة تحولات مجموعة التحكم. إن ما يميز بين برنامج الإصدار الصغير من برنامج الإنترنت هو أنه لا يمكن أن ينقطع وأنه مخصص لمهمة واحدة. وبناء على ذلك، يملك هذا البرنامج ذاكرة تتحول من حالة إلى أخرى.

إن أساس تجهيز برنامج الإصدار الصغير هو استبدال إحدى وسائط الإدخال بأخرى ـ على سبيل المثال، استبدال ملف تسلسلي من سجلات ذات تنسيق ثابت بقائمة انتظار رسائل وثائق XML. يبقى منطق البرنامج كما هو، وتبقى محتويات بيانات الإدخال كما هي، النموذج فقط هو ما يتم تبديله.

تُحكم البرامج الفرعية بواسطة عوامل تغيير خاصة بها. اعتماداً على الوسيطات يتم حساب نتائج معيّنة وإعادتها. إن منطق البرنامج الفرعي هو فعلاً من الخادم الذي ينفذ طلبات مرسلة من التابع ـ المستدعي. يجب أن يعامل كل طلب استدعاء مستقلاً عن الاستدعاءات الأخرى. إذا لم يكن البرنامج يحتفظ بالحالة، فيكون ذلك لهدف معيّن ـ ذلك بهدف استمرار المعالجة عند نقطة محددة كما في حالة عكس البرنامج[15].

هذا يجعل تجهيز البرامج الفرعية أسهل نسبياً. يحتاج المرء إلى محاكاة الواجهة. يبقى سلوك الوحدة ثابتاً. حتى أنه يمكن تحقيق الاستدعاء المرجعي عن طريق تخزين العوامل المتغيرة في البرنامج التجهيز وتمرير مرجعيات العنوان إلى إجراء هدفي. إذا استخدم نوع واجهة آخر، كوثيقة بصيغة XML في موقع في هيكلية بيانات كوبول، يجب أن يتم تحويل الواجهة الجديدة إلى واجهة قديمة قبل استدعائها من قبل البرنامج الفرعي. عند استعادة الواجهة القديمة (مثلاً، هيكلية كوبول) يتم تحويلها مرة أخرى إلى واجهة جديدة (مثلاً، وثيقة بصيغة XML). هنا تنطبق تقنيات API قياسية[42].

8 ـ 5 ـ 1 تجهيز برامج الإنترنت بواجهة بصيغة XML

تستجيب البرامج التي تعمل من خلال الإنترنت إلى طلبات التزامن من

المواقع الإلكترونية للمستخدمين، وهذا يعني أن المستخدم يملأ الصفحة الإلكترونية ويرسلها وينتظر الاستجابة. يجب أن تستجيب برمجية الجهاز الخادم بسرعة حتى لا يضطر المستخدم إلى الانتظار لفترة طويلة، وهذا يعني ضمنياً أنه يجب أن يكون هناك مسارات قصيرة للمعاملات، وأن يكون خالياً من العمليات والبيانات غير الضرورية. يجب أن يتم ضبط عملية تجهيز البرامج التي تعمل من خلال الإنترنت لتلبية هذه الاحتياجات. عملية SoftWrap الموضحة هنا تتكون من خمس خطوات تعمل كل منها حسب الحالة التي تتركها الخطوة السابقة لها:

- أولاً: يجرّد البرنامج من جميع عمليات TP-Monitor الموجودة، حيث يتم حذفها واستبدالها باستدعاءات كوبول قياسية لبرنامج التجهيز.

- ثانياً: يجرّد البرنامج من جميع كتل الشيفرة البرمجية غير المطلوبة لتنفيذ المعاملات المخصصة من قبل المستخدم.

- ثالثاً: يتم نقل كافة البيانات التي كانت تنتمي مسبقاً إلى خرائط الإدخال/ الإخراج، أو التي جُعلت قابلة للوصول للبرنامج من خلال مواقع التخزين المشتركة، وذلك إلى هيكلية بيانات واحدة في قسم الربط.

- رابعاً: يتم تحليل استخدام البيانات في قسم الربط لتحديد ما إذا كانت بيانات إدخال أو إخراج أو كليهما معاً.

- خامساً، يتم تحويل هيكلية بيانات الربط إلى جدول وصف بيانات بصيغة XML لاستخراج قيم البيانات من نماذج XML الواردة وتحويلها إلى هيكلية بيانات بلغة كوبول ولإنتاج وثيقة مخرجات بصيغة XML من الحقول المستخدمة كمخرجات.

إن سبب نزع عمليات CICS وIMS-DC هو تقليل معالجة الخريطة ولجعل البرنامج مستقلاً عن ملكية برنامج مراقب معالجة المعاملات TP Monitor(*).

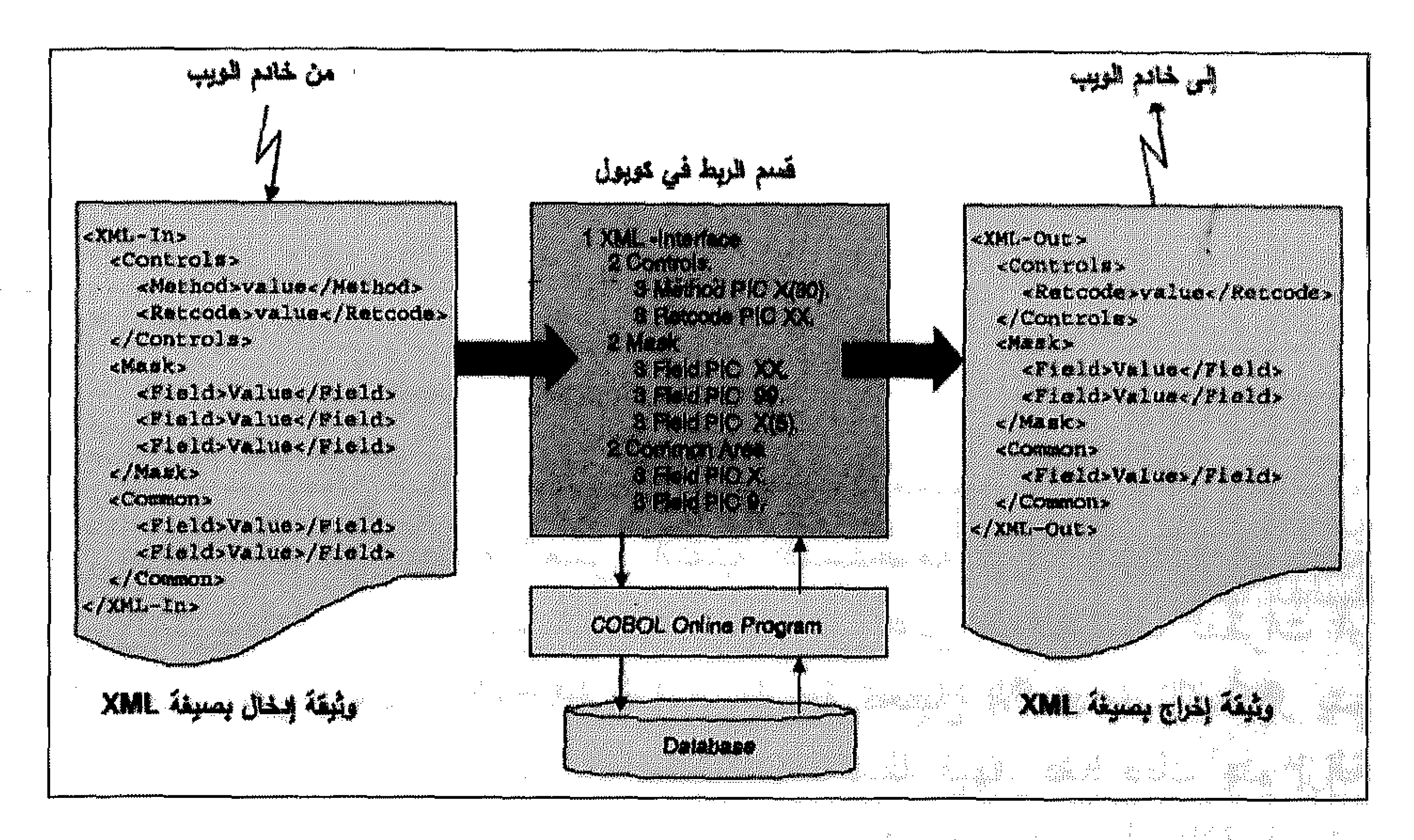
(*) نظام معالجة المعاملات أو مراقب معالجة المعاملات هو مجموعة من المعلومات التي تعالج معاملات البيانات في نظام قاعدة البيانات الذي يراقب برامج المعاملات (نوع خاص من البرامج). أما ماهية برنامج المعاملات فهو أنه يدير البيانات التي يجب أن تبقى في حالة متسقة وثابتة؛ على سبيل المثال، إذا أجريت معاملة الدفع الإلكتروني، يجب أن يتم سحب المبلغ المدفوع من حساب وإضافته إلى حساب آخر؛ فالبرنامج لا يمكن أن يكمل واحدة من هاتين الخطوتين فقط (المترجم).

البرامج التي كانت محكومة بأحداث CICS أصبحت الآن محكومة بالبيانات[27].

إن السبب المنطقي وراء نزع الشيفرة البرمجية غير الضرورية هو التخلص من التكرار. يكرّس جزء كبير من الشيفرة البرمجية لبرامج معالجة المعاملات التي تتم من خلال الإنترنت لأغراض الإدارة من أجل مراقبة المعالجة التي تتم عن بعد. يمكن حذف معظم هذه الشيفرة البرمجية، إن لم يكن جميعها، من دون التأثير في منطقية العمل. التقنية المستخدمة لإزالة الشيفرة البرمجية هي التشريح الإجرائي. يقوم المستخدم بالتأشير على الوظائف - الفقرات أو الأقسام - التي يرغب بالاحتفاظ بها. وبواسطة تحليل الاستدعاء المتكرر يتم التأشير على جميع الفقرات والأقسام المستخدمة فيها. بعد ذلك يتم إزالة جميع الشيفرة البرمجية غير المؤشرة، بما في ذلك الفقرات و/أو الأقسام. تم تغطية تقنية التشريح الإجرائي وتطبيقها في إزالة الشيفرة البرمجية في المادة المطبوعة[14].

إن جمع البيانات لإعادة هيكلتها كعوامل متغيرة أمر مهم، وذلك لأنه لم يعد برنامج مراقب معالجة المعاملات يوفّر الخرائط بعد الآن. الآن يتم تمرير بيانات الإدخال كعوامل متغيرة إلى برنامج التجهيز الذي يقوم بدوره بتمرير بيانات الإخراج رجوعاً إلى المستدعي كعوامل مخرجات. وهذا يعني أن عناصر البيانات التي كانت سابقاً محتواة في الخرائط يجب أن يتم تحويلها الآن إلى قسم الربط، لكن ليس جميعها. الخرائط تضمّنت أيضاً عدداً من حقول التحكم - على سبيل المثال، خاصية طول الكلمة بوحدة البيانات التي لم تعد ضرورية. يتم حذفها من هيكلية البيانات. إضافة إلى ذلك، يتم حذف جميع بيانات الإدارة غير المرتبطة بمهام عمل أيضاً، وهذا يترك البيانات التي يتم الرجوع إليها فعلياً من قبل البرنامج الجديد المختصر. أظهرت التجربة أن هذا يؤدي إلى تقليل حجم قسم بيانات البرنامج بنحو 50 في المئة[13].

احتفظ برنامج التجهيز الذي يعمل من خلال الإنترنت بهذه العوامل الثلاثة - معرّف العملية (Method Id)، وشيفرة الترجيع (Return Code)، وهيكلية البيانات (Data Structure)، المرتبطة بربط وثائق XML.



الشكل (8 ـ 1): تجهيز برامج الإنترنت

معرّف العملية مطلوب إذا رغب المستخدم باستدعاء وظيفة محددة من وظائف البرنامج من دون المرور بالوظائف الأخرى. إذا لم يكن معرّف العملية معداً، يتم تنفيذ البرنامج كاملاً. أما شيفرة الترجيع فهي مطلوبة لإعادة حالة الخطأ إلى الجهاز التابع في حال وجود خطأ. أما معالجة الاستثناءات فتُترك إما للتابع أو لبرنامج التجهيز (انظر الشكل 8-1).

تتضمن هيكلية البيانات كلاً من العوامل المتغيرة من موقع التابع الإلكتروني والنتائج الناجمة عن الخادم. إذا لم تكن العوامل المتغيرة قد حُررت بعد من قبل التابع، فيجب التحقق منها مرة أخرى قبل معالجتها وفقاً لمجموعة من القواعد المنطقية ـ الشروط المسبقة. كما يجب أن يتم التحقق من نتائج برنامج الخادم أيضاً للتأكد من مطابقتها لمجموعة من الشروط اللاحقة قبل إعادتها إلى المستدعي[23].

للتمييز بين العوامل المتغيرة والنتائج، يتم إجراء تحليل لاستخدام البيانات للشيفرة الإجرائية. يتم التأشير على البيانات إما كبيانات إدخال أو إخراج أو إدخال/ إخراج. يتم تحديد ذلك مقابل إنشاء هيكليتي بيانات منفصلتين، واحدة للإدخال وأخرى للإخراج، لأن هناك العديد من العوامل التي تكون من كلا النوعين، إدخال وإخراج، وهذا قد يؤدي إلى تعارض بعض الأسماء في الشيفرة البرمجية. أثبت ذلك أنه من الأفضل وجود هيكلية بيانات واحدة لكن

لتعليم عناصر البيانات الأولية وفقاً لاستخدامها. هذا يتيح استخدام هيكلية البيانات نفسها للإدخال والإخراج. طبعاً، لا يزال هناك وثيقتا XML منفصلتين، إحداهما تلك التي يتم تهيئة هيكلية بيانات كوبول منها، والأخرى هي تلك التي يتم إنشاؤها من هيكلية بيانات كوبول. أما الإجراء المتبع لإنشاء جدول وصف بيانات XML فهو مشترك بين برامج التي تعمل من خلال الإنترنت والإصدارات الصغيرة، ويتم وصفها لاحقاً في قسم منفصل.

بعد إكمال العملية ذات الخطوات الخمس، هناك نسخة التجهيز للبرنامج الذي يعمل من خلال الإنترنت السابق، حيث يمكن تجميع الإصدار وتحميله ليتم تنفيذه في بيئة الإنترنت. يبقى الإصدار الأصلي على حاله من دون تغيير في بيئة CICS أو IMS. إذا حدث تغيير منطقي على البرنامج الأصلي، فإن نسخة البرنامج المشتقة الخاصة بالإنترنت يجب أن يعاد إنشاؤها. لهذا السبب يجب أن تكون عملية التجهيز مؤتمتة. كل ما على المبرمج المسؤول عمله هو بدء العمل المؤتمت لإنشاء نسخة الإنترنت وتجميعها وربطها.

8 ـ 5 ـ 2 تجهيز البرامج الفرعية بواجهة XML

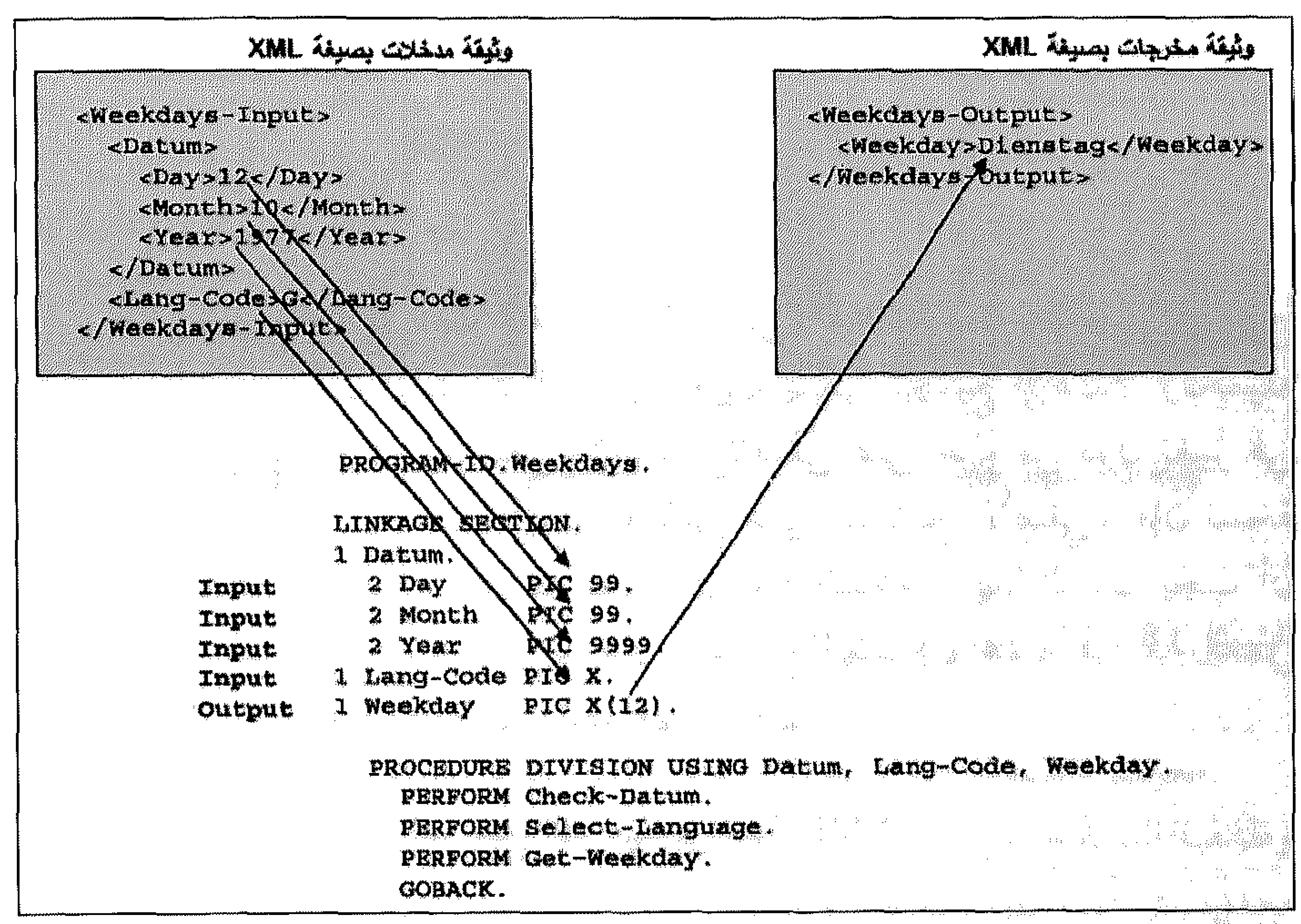
البرامج الفرعية التي تعمل ضمن بيئة معالجة المعاملات التي تتم عن طريق الإنترنت تكون أكثر عرضة للتجهيز بما إنها تمتلك واجهة وتحكمها البيانات. المهمة الوحيدة هنا هي فصل عوامل الإدخال المتغيرة (أي المتغيرات) عن عوامل الإخراج (أي النتائج). إذا كان أحد العوامل عامل مدخلات وإخراج معاً، يجب أن يؤشر عليه بناءً على ذلك. إذا كان العامل مؤشراً عليه كعامل مخرجات، يتم استخراجه وإدخاله في وثيقة الإخراج. إذا كان عامل مدخلات وإخراج معاً، يتم إعداده من وثيقة الإدخال وإدخاله إلى وثيقة الإخراج (انظر الشكل 8-2).

وبالتالي، هناك خطوتان تتعلقان بعملية تجهيز البرنامج الفرعي:

- أولاً، لتحديد كيف تستخدم عوامل الربط عن طريق تحليل مجموعاتها واستخداماتها.
- ثانياً، لإنشاء وصف بيانات XML لبدء واستخراج العوامل.

إن استدعاء روتين شيفرة تحويل العوامل XMLIN لقسم الربط كاملاً هي مهمة برنامج التجهيز، حيث يتم ذلك من وثيقة مدخلات XML. بعد إعادة التحكم من البرنامج الفرعي، يتم استدعاء روتين شيفرة تحويل العوامل

XMLOUT لإنشاء ملف إخراج XML من حقول المخرجات في قسم الربط. هناك برنامج تجهيز قياسي يدعى XML Wrap يستخدم لهذه الغاية.



الشكل (8 ـ 2) : تجهيز البرامج الفرعية

8 ـ 5 ـ 3 تحويل XML إلى كوبول وبالعكس

إن أساس تحويل بيانات XML إلى بيانات كوبول هو جدول وصف بيانات XML المنتج من وصف بيانات كوبول. لكل هيكلية كوبول، يتم إنشاء مدخل مجمع بعلامات بداية ونهاية. ولكل حقل من الحقول الابتدائية في لغة كوبول، يتم إنشاء علامة حقل ذات ست خصائص (انظر الشكل 8-3):

<level> : = رقم مكوّن من خانتين

<name> : = اسم بلغة كوبول مكوّن من 30 رمزاً

<type> : = نوع البيانات بلغة كوبول مكوّن من رمز واحد

<pos> : = موقع الحقل بمقدر بالبايت من بدايته

<lng> : = طول الحقل مقدر بعدد الرموز

<occurs> : = عدد مرات وجود الحقل

< defines > : = حقل قد يعيد تعريفه الحقل الحالي

< usage > : = استخدام الحقل إدخال/ إخراج/ إدخال إخراج

عند تحويل بيانات XML إلى كوبول، يتم التحقق أولاً من جدول وصف البيانات من حيث نوع الوثيقة المعرفة لضمان أن هيكلية البيانات صحيحة[43]. إن لم تكن صحيحة، يتم رفض الوثيقة. وبخلاف ذلك، يتم تحويل جدول وصف البيانات إلى جدول داخلي من خلال وحدة شيفرة تحويل العوامل ويتم تحديد منطقة تخزين سعتها كسعة آخر موقع زائد آخر طول. ومن ثم يُقرأ ملف محتوى XML؛ يتم التحقق من كل بند ومطابقته بالجدول الداخلي، يتم تحويل قيمته وتخصيصها إلى الموقع المناسب ضمن مساحة التخزين المحددة، علماً بأن استخدامها سيقتصر على الإدخال أو الإخراج. يتم تعيين حقول البيانات غير المشار إليها في وثيقة XML إلى القيمة الافتراضية كفراغات أو أصفار.

XML Data Description	COBOL Data Structure
<Linkage-Section>	LINKAGE SECTION.
<Parameter>	
<Field>	1 Datum.
<Level>1</Level>	
<Name>Day</Name>	
<Type>S</Type>	
</Field>	
<Field>	2 Day PIC 99.
<Level>2</Level>	
<Name>Day</Name>	2 Month PIC 99.
<Type>9</Type>	
<Pos>1</Pos>	2 Year PIC 9999.
<Lng>2</Lng>	
<Occurs>1</Occurs>	
<Usage>Input</Usage>	
</Field>	
....	
<Field>	1 Lang-Code PIC X.
<Level>1</Level>	
<Name>Lang-Code</Name>	
<Type>X</Type>	
<Pos>9</Pos>	
<Lng>1</Lng>	
<Occurs>1</Occurs>	
<Usage>Input</Usage>	
</Field>	

الشكل (8 ـ 3): تحويل بيانات XML/ كوبول

عند تحويل بيانات كوبول إلى XML، يتم إنشاء جدول وصف البيانات الداخلي، وتبدأ وثيقة XML بترويسة وصفحة نمطية. ومن ثم يتم معالجة

جدول وصف البيانات بطريقة متسلسلة لتحديد جميع الحقول المؤشر عليها كحقول مدخلات أو مخرجات. لكل حقل من حقول المخرجات، القيمة الموجودة في الموقع المشار إليه في الجدول تحوّل من النوع المشار إليه في جدول وصف البيانات ويتم إدخالها كقيمة حرفية مع علامة اسم ملائمة في وثيقة مخرجات XML. ثم تعاد وثيقة المخرجات إلى التابع.

تنطبق هيكليات بيانات كوبول الهرمية تماماً مع هيكليات بيانات XML الهرمية، بما أن كوبول تستخدم صيغ حقول ثابتة، فإن تحويلها من وإلى سلاسل برموز شيفرة ASCII أمر سهل. إن هيكليات بيانات XML وكوبول متوافقة تماماً. في الواقع، يبدو الأمر كما لو أن هيكليات بيانات كوبول وهيكليات بيانات XML من مصدر واحد[8].

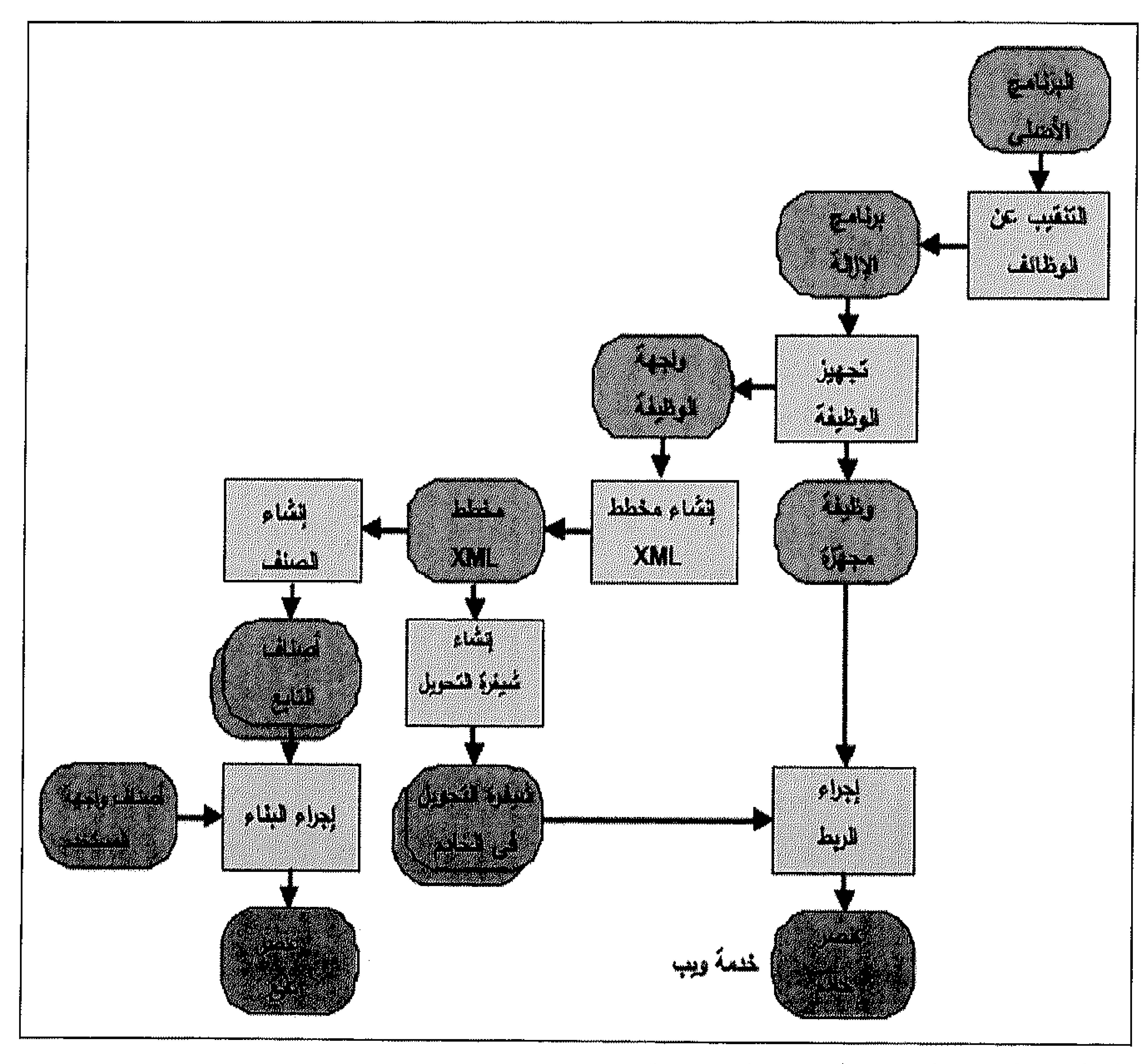
8 ـ 5 ـ 4 عملية الأداة

تتكون عملية الأداة (SoftLink) التي طورها المؤلف لربط المكوّنات المتوارثة مع الويب من سبع خطوات:

- الخطوة الأولى: خطوة التنقيب عن الوظيفة. يتم التأشير على الوظائف التي سيعاد استخدامها وتستخرج من البرنامج الموجود مع البيانات التي تستخدمها.
- الخطوة الثانية: خطوة التجهيز. يتم إنشاء واجهة جديدة لكل وظيفة مستخرجة.
- الخطوة الثالثة: خطوة إنشاء XML. يتم إنشاء مخطط فرعي بصيغة XML من وصف الواجهة في لغة البرمجة الأصلية للبرنامج.
- الخطوة الرابعة: إنشاء شيفرة تحويل العوامل للخادم لتحليل رسائل XML الواردة وترتيب الرسائل الصادرة.
- الخطوة الخامسة: خطوة تحويل التابع. يتم إنشاء أصناف جافا (Java Classes) من المخطط الفرعي بصيغة XML وذلك لإرسال واستقبال رسائل XML.
- الخطوة السادسة: خطوة الربط بالخادم. يتم ربط جميع كتل «شيفرة تحويل العوامل stubs» مع وظائف الخادم في المستضيف وذلك لإنشاء dlls.

- الخطوة السابعة: خطوة بناء التابع. يتم بناء أصناف جافا المتكونة في مكوّن واجهة ربط جافا، وذلك لربط الموقع الإلكتروني مع خدمات الويب في المستضيف (انظر الشكل 8-4).

التنقيب عن الوظائف. في خطوة التنقيب عن الوظائف، تعرض الأداة SoftWrap مصدر البرنامج المتوارث الذي تستخرج منه الوظائف لإعادة استخدامها كخدمات ويب. في حال استخدام لغة التجميع، فقد تختار المستخدم CSECTS أو علامات تمييز مفردة. أما في حال وجود PL/I فقد يختار المستخدم الإجراءات الداخلية أو الكتل المتداخلة. وأما في حال استخدام لغة كوبول فقد يختار المستخدم الأقسام أو الفقرات. وأما في حال استخدام C فقد يختار المستخدم الوظائف أو الإجراءات. في كل الأحوال، يتم استخراج وحدة الشيفرة البرمجية المختارة معاً مع جميع الوحدات التابعة ـ أي أنه يتم الإشارة إلى برامج الروتين الفرعية بتلك الوحدة.



الشكل (8 ـ 4): عملية إنشاء خدمة ويب

بما إن البرامج الروتينية الفرعية قد تشير إلى برامج روتينية فرعية أخرى، فإن عملية إرفاق شيفرة برمجية مستقلة تُكرر إلى أن يتوقف العثور على المزيد من البرامج الروتينية الفرعية. وهذا يقابل التشريح الإجرائي وهو خوارزمية تكرارية لجمع نقاط المخطط الإجرائي الرسومي الذي عُرض في ورقة بحث سابقة[31]. إذا تم جمع العديد من البرامج الروتينية الفرعية، فقد تصبح خدمة الويب النظرية كبيرة جداً ويجب إهمالها. لذا، من الأفضل دراسة تأثير مجال عمل الوظيفة المختارة قبل استخراجها. كما أنه من الضروري الحد من تفرعات Go To في المجال المختار لضمان أن الوظيفة يمكن أن تُنفّذ باستقلالية.

تجهيز الوظيفة. حالما يتم استخراج الوظيفة من المصدر الموجود مع الوظائف الفرعية التابعة لها، تكون الخطوة التالية تجهيز الوظيفة. يتم ذلك أوتوماتيكياً باستخدام الأداة SoftWrap التي تُنشئ مدخلاً وواجهة لتلك الوظيفة. يسبق عملية إنشاء الواجهة تحليل لتدفق البيانات التي تحدد جميع المتغيرات المشار إليها من قبل الوظيفة سواء كان ذلك مباشرة أو بشكل غير مباشر. ويتضمن ذلك هيكليات البيانات الأساسية والمتجهات المفهرسة. إذا كان أحد العناصر الأولية في الهيكلية مشاراً إليه، فإن جميع الهيكلية ستكون مضمّنة في الواجهة.

في النهاية، تصبح كافة البيانات المستخدمة في وظيفة التجهيز جزءاً من واجهتها. إذا كانت الوظيفة صغيرة، ستكون الواجهة ضيقة. أما إذا كانت الوظيفة كبيرة، فإن الواجهة تصبح أوسع. ويعود تضييق نطاق الوظائف للمستخدم.

في أي حالة من الحالات، تكون الوظائف عبارة عن مكوّنات عديمة الحالة. فهي لا تحتوي أي بيانات ثابتة. وهذه الخاصية تجعل من المكوّنات قابلة لاستخدامها في المعالجة متعددة المواضيع. البيانات تكون تابعة للمهمة، في حين تكون الشيفرة البرمجية قابلة لأن تكون مشتركة.

إنشاء مخطط XML. بعد تجهيز الوظائف القابلة لإعادة الاستخدام المختارة وإنشاء الواجهات الوظيفية الجديدة، تأتي الخطوة التالية وهي تحويل هذه الواجهات إلى مخطط XML، في حين يتم في الوقت نفسه إنشاء شيفرة التحويل الخاصة بالخادم لمعالجة رسائل XML بناءً على ذلك المخطط. عندما يتم تجهيز الوظائف، يتم إنشاء واجهاتها بلغة البرمجة الأصلية سواء كانت لغة التجميع أو PL/I أو كوبول أو C. تصبح الواجهات إما ماكرو لغة التجميع أو توابع PL/I أو

نسخ كوبول أو ملفات ترويسية للغة C. بهذه الطريقة تكون الواجهات مستقلة عن الوسيط. وقد يتم ترجمتها إما إلى CORBA-IDL أو إلى XML. إلى هنا، يتم حفظ الواجهات في مكتبة الماكرو في الخادم المركزي.

تُحوّل الواجهات إلى XML لغايات ربط الوظائف المستخرجة مع الويب. كل واجهة من الواجهات هي بالضرورة هيكلية معرفة كنوع معقد من XML. أما خصائصها فهي الاسم والنوع وموقع الاسم. إذا كان هناك هيكليات فرعية ضمن الهيكلية الأصلية، فإنها ستصبح أنواعاً معقدة أيضاً. في حالة إعادة المجموعات، يكون أقصى عدد مرات التكرار معطى.

العوامل المفردة هي عناصر ذات خصائص قياسية متوقعة مسبقاً من قبل XMI، وهي تحديداً Name وType وHref وOccurs وغيرها. إضافة إلى ذلك، هناك خصائص ممتدة مخصصة لتسهيل تحويل البيانات إلى أنواع EBCDIC في المستضيف. وهي كما يأتي:

- Pos := إزاحة حقل البيانات ضمن هيكلية بيانات المستضيف.
- Lng := طول حقل بيانات المستضيف مقدراً بالبايت.
- Pic := نمط نوع التحرير بلغة كوبول أو PL/I. مثلاً S999.99.
- Use := الاستخدام، مثلاً حسابي، عرض وغير ذلك.

تتيح هذه الخصائص لأداة التحليل وضع بيانات المدخلات في الموضع الملائم، بالطول الملائم وبصيغة ملائمة ضمن حاجز(*) الرسائل التي تتضمن عوامل الوظيفة المستدعاة. عكسياً، قد تستخرج بيانات المخرجات من حاجز الرسائل لوضعها في وثيقة XML صادرة. مخطط XML هذا هو أساس جميع عمليات المعالجة الإضافية. حالما يتم إنشاؤها، يتم تخزينها في موقع تخزين XML.

إنشاء شيفرة التحويل للخادم. في الخطوة الرابعة من مخططات XML، يتم إنشاء أدوات تحليل معالجة رسائل XML الواردة من التوابع وترتيب عودتها إلى التوابع. بهدف التوافق مع نظام وقت التشغيل الذي يتم تشغيل

(*) العازل أو الحاجز (Buffer) هو مكان مؤقت في الذاكرة حيث يتم فيه تخزين البيانات حين تنقَل من مكان إلى آخر (المترجم).

الوظائف المستهدفة فيه، يتم إنشاء شيفرة التحويل لأداة التحليل بلغة البرمجة نفسها التي استخدمت لبناء الوحدات الوظيفية. في حال استخدام لغة كوبول، تبرمج شيفرة التحويل لأداة التحليل بلغة كوبول، أما في حال استخدام PL/I فتبرمج بهذه اللغة، وهكذا. وهذا من شأنه أن يتجنب المشكلات الناجمة عن التواصل عبر اللغات.

عند بدء العملية، يتم قراءة مخطط XML من موقع تخزين XML ويتم إنشاء جدول وصف البيانات الداخلية مع مدخل لكل عامل من العوامل. يتضمن المدخل الاسم والصورة والاستخدام والإزاحة والطول وتكرار الحدوث. ومن ثم عند استدعاء الوظيفة المستهدفة، فإنها تستدعي شيفرة تحويل المدخلات لإعطائها للرسالة الواردة. تقرأ شيفرة التحويل وثيقة XML التالية من طابور الإدخال، وتلتقط قيم البيانات من الوثيقة وتحولها إلى أنواع بيانات المستضيف. بعد ذلك يتم تخصيص القيم المحولة إلى الحقول المقابلة في الفراغ المخصص للعنوان في برنامج التجهيز.

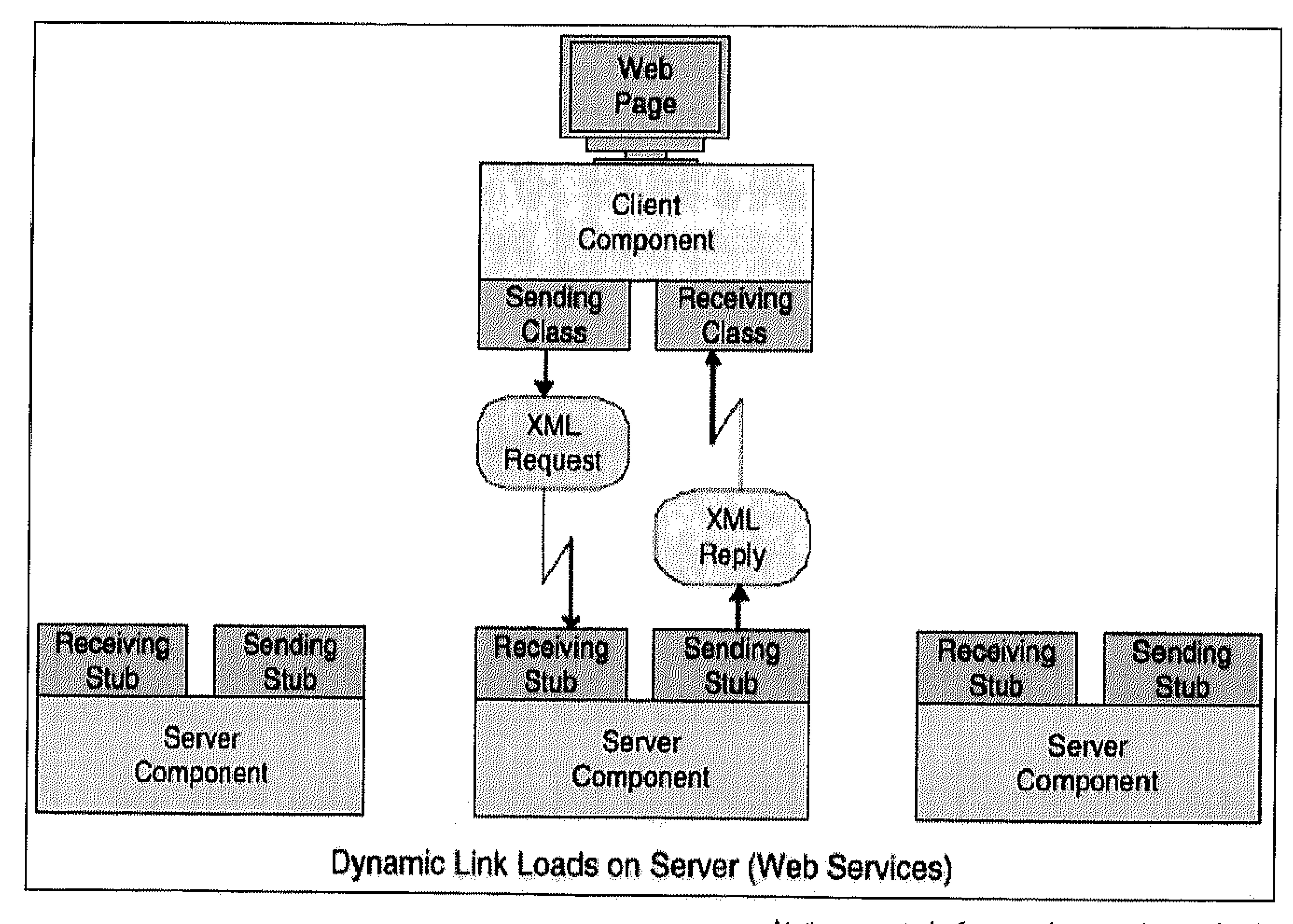
يتم تشغيل شيفرة تحويل المخرجات عندما يستلزم الأمر كتابة خريطة أو سجل أو تقرير، أو عند الخروج من وظيفة التجهيز. في أي حالة، تأخذ شيفرة التحويل نتائج بيانات المخرجات من الفراغ المخصص للعنوان في البرنامج المجهّز وتحولها إلى قيم رموز ASCII وتدخلها في وثيقة إخراج XML لتحويلها إلى التابع. إن الاحتفاظ بجدول وصف البيانات الداخلي في ذاكرة الاحتفاظ في الخادم، لن يلزم مخطط XML سوى تفسيره مرة واحدة فقط وذلك في بداية كل عملية.

إنشاء صنف التابع. مخططات XML التي تعتبر أساس إنشاء شيفرة التحويل في الخادم تستخدم نفسها لإنشاء أصناف التابع. لكل وظيفة في الخادم (أي خدمات الويب)، يتم إنشاء صنفين اثنين من نوع جافا، أحدهما لإنشاء الطلب لوظيفة الخادم، والآخر لتفسير النتيجة الناتجة من الخادم. تستدعى عمليات النوع من قبل أصناف واجهة المستخدم التي تعالج صفحة والويب الخاصة بالمستخدم.

هذان الصنفان الناتجان لهما غرض مضاعف. فمن ناحية، يحفظان مبرمجي التابع من الوقوع في مشكلة كتابتهما، ومن ناحية أخرى يضمنان أن تكون واجهات XML بالنسق نفسه.

ربط خدمات الويب. في الخطوة الأخيرة، يتم ربط وظائف الخادم معاً مع وحدات تحميل رابط ديناميكي مستقل ليكون متاحاً في الجهاز المستضيف، الذي قد يعمل ضمن WebSphere أو WebLogic. تعمل وحدات dll بطريقة خدمات الويب نفسها ويمكن أن تكون متوفرة لـ DotNet أيضاً. يمكن استدعاؤها من أي متصفح إنترنت من أي مكان في الشبكة. يجب أن يتضمن استدعاء عنصر التابع الأنواع (Classes) لإرسال واستقبال واجهات XML إلى وظائف الخادم. بطبيعة الحال بالنسبة إلى أنواع DotNet، يتم إنشاء الأنواع بلغة C++ لغايات التوافقية.

الفارق المهم بالنسبة إلى خدمات الويب المقترحة من مايكروسوفت هو حقيقة أن هذه الخدمات تنتج أوتوماتيكياً من البرامج القديمة الموجودة من دون تدخل يدوي. وهذا يميزها من أساليب إعادة التصميم المعتادة. فهي تمثل توفيراً مهماً في جهود التصميم والبرمجة والاختبار. إضافة إلى ذلك، فهي تلبّي متطلبات إعادة استخدام وظائف البرامج القديمة المتطلبة منذ زمن من دون الحاجة إلى إعادة برمجتها. يتم الوصول إلى وظائف التطبيق الموجودة من خلال الإنترنت، كما تصورها كل من تيبت Tibbetts و Bernstien (انظر الشكل 8-5)[38].



Dynamic Link Loads on Server (Web Services)

الشكل (8ـ 5): هيكلية خدمة الويب

8 ـ 6 الخبرة العملية

طبقت الطريقة الموصوفة أعلاه على برنامج بنكي واسع النطاق، مكتوب بلغة البرمجة كوبول، ويعمل ضمن IMS-DC مع قاعدة البيانات العلائقية DB2. شمل النظام 92 وحدة كوبول إضافة إلى 11 وحدة بلغة التجميع ارتبطت هذه الوحدات بـ 65 جدولاً في قاعدة البيانات. أما الغاية من هذا البرنامج فكان معالجة المعاملات الواردة من آلة الصرّاف الآلي ATM. كان هدف المشروع جعل نظام الكوبول قابلاً للتنفيذ في CICS وفي خادم التطبيقات WebShere من IBM. لتحقيق هذا الهدف، حُذفت عمليات IMS-DC واستبدلت بأداة تجهيز تعمل على بناء طابور للمعاملات الواردة من آلة الصراف الآلي وتغذيها بالتسلسل إلى وظائف العمل الأساسية التي تعمل بدورها على استدعاء روتين الوصول إلى قاعدة البيانات. بهذه الطريقة، أمكن حفظ كتلة الشيفرة البرمجية للغة كوبول، على الرغم من أن النظام نفسه كان مضمّناً ككائن في الهيكلية الموزعة.

كان بالإمكان استدعاء الوحدات المنفصلة من النظم الغريبة طالما أن هذه الوحدات استلمت الواجهة القياسية التي تحدد فيها البيانات والوظائف الخاصة بها، هذا بالإضافة إلى كونها قابلة للاستدعاء ككائن واحد كبير. بعد اكتمال هذا المشروع التجريبي، أثبت جدوى تجهيز نظم معالجة المعاملات عن طريق الإنترنت حتى الكبيرة منها وتجهيزها للترحيل إلى بيئة عمل جديدة. كان الأداء معتمداً على WebSphere لكن كان أداء النظام أقل من التطبيقات البنكية الأخرى التي تعمل من خلال WebSphere[29].

8 ـ 7 الاستنتاج

حددت هذه المساهمة استراتيجيات لتزويد الهيكلية خدمية التوجه بخدمات الويب. كما أشرنا مسبقاً، يمكن شراء خدمات الويب أو استئجارها أو استعارتها أو تطويرها أو استعادتها. ثمة فوائد ومساوئ لكلّ منهجية. أسرع وأرخص وسيلة لتزويد خدمات الويب، غير استعارتها من مجتمعات المصادر المفتوحة، هي استعادتها من التطبيقات الموجودة ضمن ملكية المستخدم.

ثمّة حاجة ملحة لإجراء المزيد من الأبحاث في مجال استعادة الشيفرة البرمجية. أولاً هناك حاجة للتقنيات لتحديد الشيفرة البرمجية بناءً على النتائج. ثانياً، يلزم الأمر وجود مقاييس لتقييم إمكانية إعادة استخدام الشيفرة البرمجية

المحددة. ثالثاً، يجب وجود أدوات لاستخراج سلاسل من مجموعات الشيفرة البرمجية المترابطة، أي خصائص الشيفرة البرمجية من البيئة التي تتواجد فيها تلك الشيفرة البرمجية. أخيراً، يجب توفر أدوات لتجهيز الشيفرة البرمجية المستخلصة أوتوماتيكياً وتكييفها كخدمة ويب. جميع هذه الخطوات عرضة للأتمتة؛ لكن لأتمتتها، يجب التحقق من أكثر الوسائط موثوقية، وأفضلها لتنفيذ هذه المهام، كما يجب مقارنتها بالعديد من التقنيات. في هذا الصدد، يسهم الباحثون مساهمة قيّمة في عملية الترحيل.

المراجع

1. P. Andriole. «The Colaborale Integrate Business Technoogy Strategy.» *Communications of the ACM*: vol. 49, no. 5, 2006, pp. 85-90.

2. L. Aversano [et al.]. «Migrating legacy systems to the Web is an experience report. paper presented at: *Proceedings of the 5th European Conference on Software Maintenance and Reengineering (CSMR 2001)*. Lisbon: IEEE Computer Society Press, 2001, pp. 148-157.

3. J. Benamati and A. Lederer. Coping with rapid changes in IT. *Communications of the ACM*: vol. 44, no. 8, 2001, pp. 83-88.

4. P. Bharati. India's IT service industry is a competetive analysis. *IEEE Computer*: vol. 38, no. 1, 2005, pp. 71-75.

5. M. Bichler and L. Kwei-Jay. Service-oriented computing. *IEEE Computer*: vol. 39, no. 3, 2006, pp. 99-101.

6. J. Bishop and N. Horspool. Cross-platform development: Software that lasts. *IEEE Computer*: vol. 39, no. 10, 2006, pp. 26-35.

7. T. Bodhuin, E. Guardabascio, and M. Tortorella. Migrating COBOL systems to the WEB by using the SOAP. paper presented at: *Proceedings of the 9th Working Conference on Reverse Engineering (WCRE-2002)*, IEEE Computer Society Press, Richmond, 2002, pp. 329-388.

8. N. Bradley. *The XML Companion: Document Modelling*. Harlow, G.B: Addison-Wesley, 2000, pp. 71-89.

9. G. Canfora [et al.]. «Migrating interactive legacy system to Web services.» *Proceedings of the 10th European Conference on Software Maintenance and Reengineering*. Bari: IEEE Computer Society Press, 2006, pp. 24-36.

10. N. Carr. «Software as a commodity.» *Harvard Business Review*: vol. 51, no. 5, May 2003.

11. P. Duchessi and I. Chengalur-Smith. «Client/server benefits, problems, best practices.» *Communications of the ACM*: vol. 41, no. 5, 1998, pp. 87-94.

12. R. Evaristo, K. Desouza, and K. Hollister. «Centralization Momentum: The pendulum swings back again.» *Communications of the ACM*: vol. 48, no. 2, 2005, pp. 67-71.

13. R. Fanta and V. Rajlich. «Removing clones from the code.» *Journal of Software Maintenance*: vol. 11, no. 4, 1999, pp. 223-244.

14. K. B. Gallagher and J. R. Lyle. «Using program slicing in software maintenance.» *IEEE Transactions on Software Engineering*: vol. 17, no. 8, 1991, pp. 751-761.

15. D. Garlan, R. Allen, and J. Ockerbloom. «Architectural mismatch: Why reuse is so hard.» *IEEE Software*: vol. 12, no. 6, 1995, pp. 17-26.

16. R. Glass. «The realities of software technology payoffs.» *Communications of the ACM*: vol. 42, no. 2, 1999, pp. 74-79.

17. O. Greevy, S. Ducasse, and T. Girba. «Analyzing Software Evolution through Feature Views.» *Journal of Software Maintenance & Evolution*: vol. 18, no. 6, 2006, pp.425-456.

18. M. Hammer and J. Champy. *Reengineering the corporation: A manifest for business revolution.* New York: Harpers Business, 1993.

19. E. Horowitz. «Migrating software to the World Wide Web.» *IEEE Software*: vol. 15, no. 3, 1998, pp. 18-21.

20. M. A. Jackson. *Principles of Program Design.* London: Academic Press, 1975, pp. 67-82.

21. D. Krafzig, K. Banke, and D. Schama. *Enterprise SOA*, Coad Series, Upper Saddle River, NJ: Prentice-Hall, 2004.

22. G. Larsen. Component-based enterprise frameworks, *Communications of the ACM*: vol. 43, no. 10, 2000, pp. 24-26.

23. B. Meyer. «Applying design by contract.» *IEEE Computer*: vol. 25, no. 10, 1992, pp. 40-51.

24. H. Q. Nguyen, B. Johnson, and M. Hackett. *Testing Applications on the Web*. Indianapolis, IN: John Wiley & Sons, 2003.

25. P. Pak-Lok and A. Lau. «The present B2C implementation framework.» *Communications of the ACM*: vol. 49, no. 2, 2006, pp. 96-103.

26. R. Seacord, D. Plakosh, and G. Lewis. *Modernizing Legacy Systems*. Reading, MA: Addison-Wesley, 2003.

27. A. Sellink, C. Verhoef, and H. M. Sneed. «Restructuring of COBOL/CICS legacy systems.» paper presented at: *Proceedings of the 3rd European Maintenance and Reengineering (CSMR 1999)*, Amsterdam: IEEE Press, 1999, pp. 72-82.

28. H. Sneed. «Measuring reusability of legacy software systems.» *Software Process*: vol. 4, no. 1, 1998, pp. 43-48.

29. H. M. Sneed. *Objektorientierte Softwaremigration*, Bonn: Addison-Wesley, 1999, pp. 1-29.

30. H. M. Sneed. «Generation of stateless components from procedural programs for reuse in a distributed system.» paper presented at: *Proceedings of the 4th European Maintenance and Reengineering (CSMR 1999)*, Zürich: IEEE Press, 2000, pp. 183-188.

31. H. Sneed. «Extracting business logic from existing COBOL programs as a basis for reuse.» paper presented at: *Proceedings of the 9th International Workshop on Program Comprehension (IWPC 2001)*, Toronto: IEEE Computer Society, 2001, pp. 167-175.

32. H. Sneed. «Wrapping Legacy COBOL Programs behind an XML Interface.» paper presented at: *Proceedings of the 8th Working Conference on Reverse Engineering (WCRE 2001)*. Stuttgart: IEEE Computer Society Press, 2001, pp. 189-197.

33. H. Sneed. «Integrating legacy software into a service oriented architecture.» paper presented at: *Proceedings of the 10th European Conference on Software Maintenance and Reengineering (CSMR 2006)*. Bari: IEEE Computer Society Press, 2006, pp. 3-14.

34. H. Sneed and K. Erdos. «Extracting business rules from source code.» paper presented at: *Proceedings of the 4th International Workshop on Program Comprehension (IWPC'96)*. Berlin: IEEE Computer Society Press, 1996, pp. 240-247.

35. H. M. Sneed and E. Nyâry. «Downsizing large application programs.» *Journal of Software Maintenance*: vol. 6, no. 5, 1994, pp. 235-247.

36. P. Strassman. «The total cost of software ownership.» *IT Cutter Journal*, vol. 11, no. 8, 1998, p. 2.

37. A. Terekhov and C. Verhoef. «The realities of language conversion.» *IEEE Software*: vol. 12, no. 6, 2000, pp. 111-124.

38. J. Tibbetts and B. Bernstein. «Legacy applications on the web.» *American Programmer*: vol. 9, no. 12, 1996, pp.18-24.

39. S. Tilley [et al.]. «On the business value and technical challenges of adapting Web services.» *Journal of Software Maintenance & Evolution*: vol. 16, no. (1-2), 2004, pp. 31-50.

40. S. Tockey. *Return on Software*. Boston, MA: Addison-Wesley, 2005.

41. A. von Mayrhauser and A. M. Vans. «Identification of dynamic comprehension processes during large scale maintenance.» *IEEE Trans. on Software Engineering*: vol. 22, no. 6, 1996, pp. 424-437.

42. D. W. Wall. «Processing Online Transactions Via Program Inversion.» *Communications of the ACM*: vol. 30, no. 12, 1987, pp. 1000-1010.

43. R. Westphal. «Strong tagging Der Ausweg aus der Interface-Versionshölle.» *Objektspektrum*: vol. 4, 2000, pp. 24-29.

9

تحليل وتصوّر تطوّر البرمجيات

مايكل فيشر (Michael Fisher)
وهارالد غال (Harald Gall)
ومارتين بنزغير (Marti Pinzger)

9 ـ 1 المقدمة

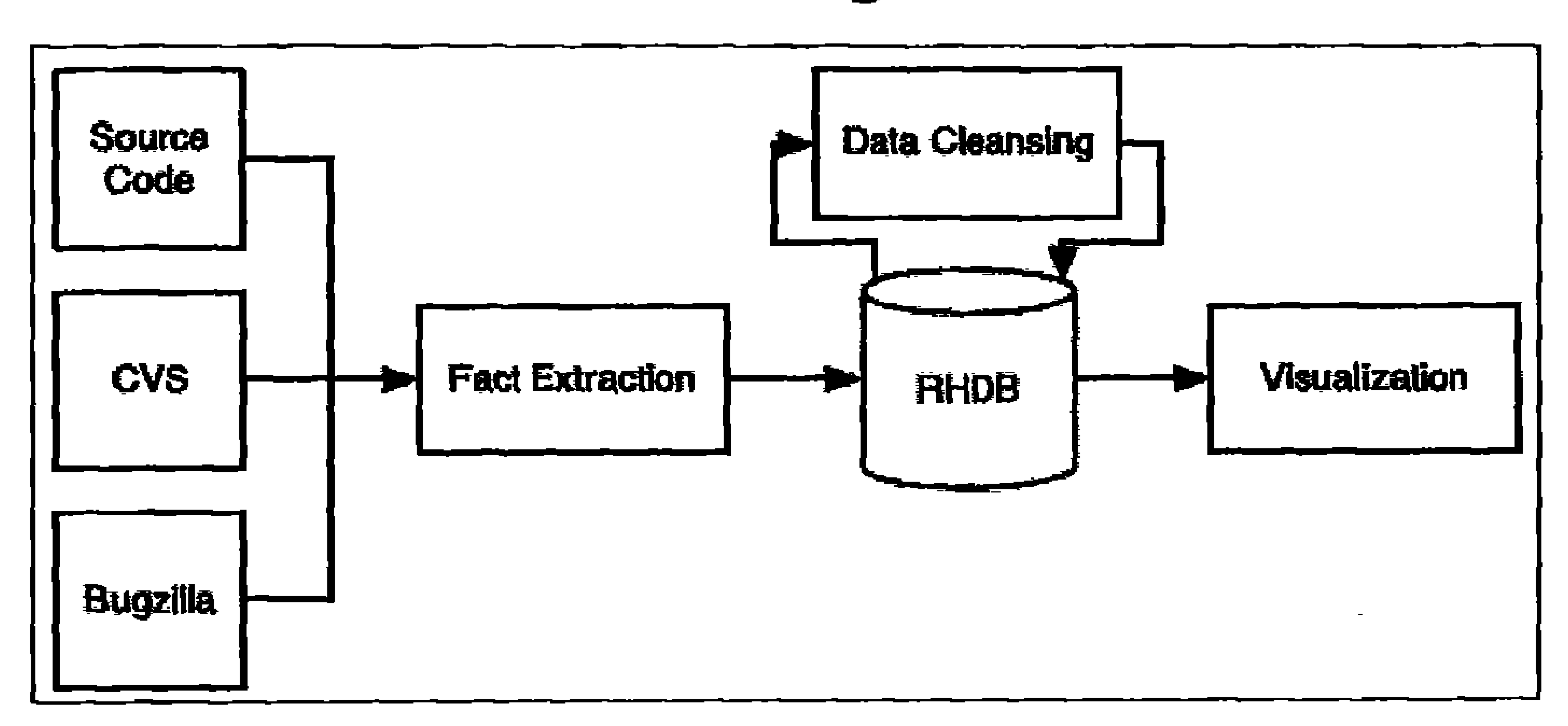
يُعنى تحليل تطور البرمجيات بتحليل التغيرات التي تطرأ على البرمجية وأسباب التغيير وتأثيره. ويتضمن ذلك تحليلاً استرجاعياً للبيانات المُخزَّنة في مواقع التخزين الخاصة بالبرمجية.

تتضمن تلك البيانات معلومات الإصدارات الماضية مع الشيفرة البرمجية المصدرية ومعلومات التغيير وبيانات الإبلاغ عن المشكلات والبيانات المستخلصة من تتبّع عمليات التنفيذ. تحديداً، اكتسبت بيانات تحليل الإصدار والإبلاغ عن المشكلات أهمية نظراً إلى كونها تخزن ثروة من المعلومات القيّمة بالنسبة إلى تحليل تطور النظم البرمجية.

أما تصور البرمجيات فهي وسيلة لدعم تحليل الجوانب المختلفة للنظام البرمجي بعروض رسومية. في هذا الفصل، نقدم أربع تقنيات تصوّر مختلفة مع التركيز على تصور جوانب نشوء البرمجيات. على وجه الخصوص، الجوانب المعنية بتطور وحدات الشيفرة البرمجية المصدرية (عرض مقاييس متعددة للتطور) وتطور الميزات (عرض تطور الميزات) ومساهمات المطوّر (عرض مساهمة المطوّر) وتقارن التغيير بين وحدات الشيفرة البرمجية المصدرية (عرض تقارن التغيير). تتيح كل تقنية للمستخدم إنشاء عروض مختلفة بهدف تسريع

تحليل وفهم النظام قيد الدراسة. على سبيل المثال، تبيّن العروض هيكلية النظام كما تم تنفيذها مدعمة بمعلومات مقاييس التطور التي تلقي الضوء على النطاقات المهمة. تشير هذه النطاقات التي نسميها بالنقاط الساخنة إلى وحدات التنفيذ غير المستقرة وعيوب تنفيذ ميزات معيّنة أو عيوب في تصميم وبناء الفريق. إن الإشارة إلى أوجه القصور هذه هي وسيلة لتحسين التصميم والتنفيذ ومشاركة الفريق، وهذا هو هدفنا الأولي.

مصدر البيانات الأساسية التي تستخدمها تقنيات التصور الأربعة المقدّمة هي قاعدة بيانات الإصدارات السابقة RHDB(7). تدمج قاعدة البيانات هذه الوقائع المستخلصة من إصدارات الشيفرة البرمجية العديدة وبيانات الميزات والإصدارات (أي CVS) وبيانات تتبع المشكلات والعيوب (مثلاً Bugzilla).



الشكل (9 ـ 1): عملية تحليل وتصور تطور البرمجية مع خطوات استخلاص البيانات (اليسار)، وقاعدة بيانات الإصدارات السابقة وخطوة تنظيف البيانات (الوسط)، ومنهجيات التصور (اليمين).

يبيّن الشكل 9-1 عرضاً عاماً لعملية تحليل وتصور تطور البرمجية. يبيّن الجانب الأيسر مصادر البيانات المختلفة، التي تؤخذ منها البيانات الأولية. حالياً، تستخدم التقنيات خاصتنا إصدارات عديدة للشيفرة المصدرية وتتبع التنفيذ، وإصدار البيانات من مواقع التخزين في CVS، وبيانات الإبلاغ عن المشكلات من مواقع التخزين في Bugzilla. تخزن الوقائع المستخلصة من مصادر البيانات المختلفة في قاعدة بيانات الإصدارات السابقة. في خطوة تنظيف البيانات، يتم ربط وقائع مصادر البيانات المختلفة ما يتيح لنا التنقل بين

المشكلات إلى توصيفات الشيفرة البرمجية المصدرية المعدّلة والميزات المتأثرة بالتغير والعكس صحيح. إضافة إلى ذلك، نحن نراعي الإصدارات العديدة ونؤسس الروابط بين الإصدارات للتنقل بين الإصدارات المختلفة لوحدة معيّنة.

بعد ذلك، ولكلّ ملف مصدر نقوم بقياس حجم مقاييس التطور ومدى تعقيد البرنامج وعدد العيوب والمشكلات المبلغ عنها وعدد التعديلات التي أجريت وغير ذلك. أما بالنسبة إلى علاقات التبعيات بين الملفات المصدرية وقوة التبعية فيتم قياسها بحساب عدد مرات استدعاء العملية وعدد مرات الوصول للخاصية وعدد التسليمات المشتركة بين ملفين. تخزن البيانات الجديدة المحصّلة في قاعدة بيانات الإصدارات السابقة، ومن هناك يمكن استرجاع البيانات باستخدام تقنيات التصور الخاصة بنا.

9 ـ 2 عرض مقاييس التطور العديدة

نقدم في هذا القسم منهجية لإنشاء عروض مختلفة ذات مستوى متقدم عن الشيفرة المصدرية. تصور هذه العروض الوحدات البرمجية وعلاقات التبعية ما بينها. تنبع الوحدات البرمجية من تحليل النظام إلى وحدات تنفيذية ذات معنى. مثل هذه الوحدات، هي عبارة عن فهارس للشيفرة المصدرية والملفات المصدرية والأنواع. تشير العلاقات التبعية إلى تبعيات الاستخدام أو توارث التبعيات. يرجع تفصيل تبعيات الاستخدام إلى العلاقات التبعية على مستوى الشيفرة المصدرية، تحديداً إلى تضمين الملف واستدعاءات العملية وصولاً إلى المتغير وتبعيات النوع.

إن الهدف من عرض مقاييس التطور العديدة هو الإشارة إلى جوانب محددة من عملية تنفيذ إحدى إصدارات الشيفرة المركزية أو أكثر من إصدار ـ على سبيل المثال، وتسليط الضوء على الوحدات التي تعتبر كبيرة ومعقدة بشكل استثنائي، والتي تفرض علاقات تبعية قوية على وحدات أخرى. إضافة إلى ذلك، يتم تسليط الضوء على الوحدات ذات الحجم ودرجة التعقيد اللتين تزيدان بقوة أو الوحدات التي تصبح غير مستقرة. يمكن استخدام مثل هذه العروض من قبل مهندسي البرمجيات، مثلاً لـ:

أ) الحصول على معلومات عن التصميم المنفّذ وتطوره.

ب) اكتشاف الوحدات المهمة التي تنفذ الوظائف الأساسية للنظام البرمجي.

ت) اكتشاف الوحدات المقترنة بقوة.

ث) تحديد توجهات التطور الحرجة.

أما الأفكار الرئيسة والمفاهيم الأساسية لعرض مقاييس التطور العديدة فقد طُورت في أعمال Pinzger والآخرين[20].

9 ـ 2 ـ 1 بيانات الشيفرة البرمجية المصدرية

بالنسبة إلى عرض مقاييس التطور العديدة، تتكون بيانات الإدخال من معلومات الشيفرة البرمجية المصدرية المهيكلة وبيانات المقاييس المستخلصة من عدد من إصدارات الشيفرة البرمجية. تحدد مقاييس الشيفرة المصدرية حجم ومدى تعقيد البرنامج كمياً وتقارن الوحدات وقوة العلاقات التبعية. إن مقياس حجم الوحدة المثالي هو عبارة عن عدد أسطر الشيفرة البرمجية وعدد العمليات وعدد الخصائص وغير ذلك. أما مقاييس درجة تعقيد البرنامج فمنها على سبيل المثال درجة تعقيد ماكابي McCabe Cyclomatic[18] ومحتوى Halstead الذكي، وجهد Halstead الذهني، وصعوبة برنامج Halstead[10]. تعطى قوة العلاقات التبعية بعدد استدعاءات العملية الثابتة أو الوصولية للخصائص بين وحدتين.

إن استخلاص حساب العلاقات التبعية وقيم القياسات تنفذ باستخدام أدوات التحليل والمقاييس. في دراسات الحالة الخاصة بنا التي تجرى على برنامج Mozilla ذي المصدر المفتوح، استخدمنا الأداة Imagix-4D[1] لحساب تحليل ومقاييس ++C/C لمجموعة مختارة من إصدارات الشيفرة المركزية. تخصص القيم القياسية لكل وحدة علاقة تبعية لمتجه إحدى الميزات. يتم تتبع متجهات الميزات خلال عدد n من الإصدارات وتكوّن مصفوفة التطور E. القيم في المصفوفة التي تُحدد تطور الوحدة أو العلاقة التبعية كمياً:

$$E_{i \times n} = \begin{pmatrix} m'_1 & m''_1 & .. & m^n{}_1 \\ m'_2 & m''_2 & .. & m^n{}_2 \\ . & . & \cdots & . \\ m'_i & m''i & .. & m^n i \end{pmatrix}$$

تتضمن المصفوفة عدد n من متجهات الميزات ذات قياسات بمقاييس i.

(1) <http://www.imagix.com> .

تحسب مقاييس التطور لكل وحدة وكل علاقة تبعية. وهي تشكّل مدخلاً أساسياً إلى منهجية التصوّر ArchView خاصتنا.

9 ـ 2 ـ 2 تصوّر قيم المقاييس المتعددة لإحدى الإصدارات

منهجية ArchView هي امتداد لتقنية العروض ذات المقاييس المتعددة المقدمة من لانزا Lanza وآخرين[16]. فبدلاً من استخدام الأشكال الرسومية ذات العدد المحدود من المقاييس القابلة للعرض، تستخدم منهجية ArchView مخططات Kiviat البيانية المعروفة أيضاً بالمخططات الرادارية البيانية[*]. هذه المخططات مناسبة لعرض قيم المقاييس المتعددة المتاحة للوحدة، كما سنبيّن لاحقاً.

يبيّن الشكل 9-2 مثالاً على مخطط Kiviat بياني يمثل قياسات ستة مقاييس هـ هي M1، M2، . . . ، M6 لإحدى إصدارات تنفيذ الوحدة moduleA. البيانات الضمنية في المخطط مأخوذة من مصفوفة التطور E:

$$E_{6x1} = \begin{pmatrix} m'_1 \\ . \\ . \\ m'_6 \end{pmatrix}$$

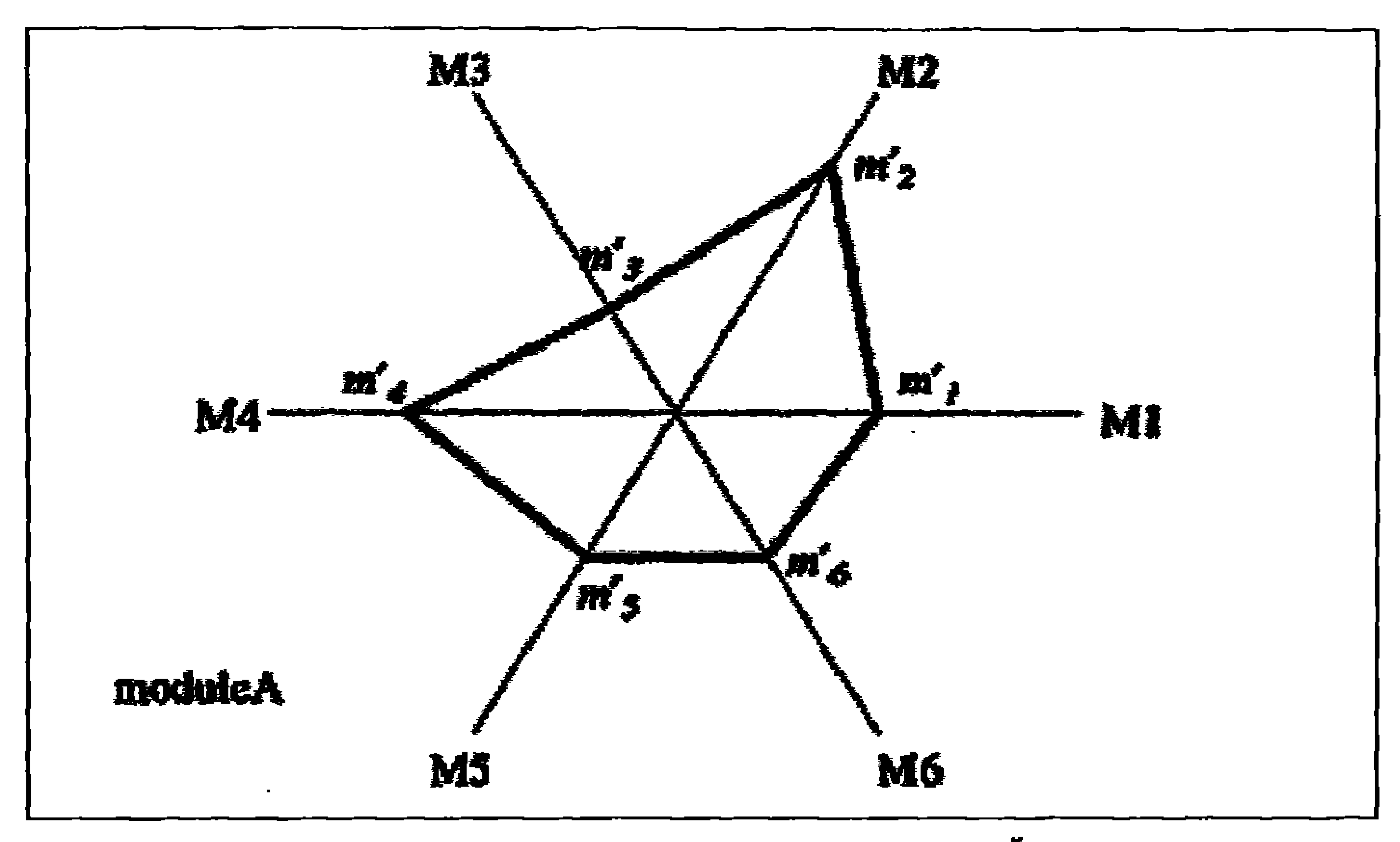
في مخطط Kiviat البياني ترتب قيم القياسات في دائرة. ثمة محور لكل قياس في المخطط. حجم المخطط ثابت وكل قيمة تسوّى حسب المخطط. في الأمثلة الواردة في هذا القسم نستخدم معادلة التسوية الآتية:

$$l(m'_i) = \frac{m'_i * cl}{\max(m'_i)}$$

حيث cl تشير لحجم مخططات kiviat والقيمة العظمى $\max(m'_i)$ هي أقصى قيمة لمقياس m'_i مستخدم في جميع الوحدات التي سيتم تصورها.

(*) مخطط الرادار البياني هو طريقة رسومية لعرض بيانات متعددة المتغيرات بصورة رسم بياني ذي بعدين لثلاثة متغيرات كمية أو أكثر ممثلة على محاور تبدأ من النقطة نفسها (المترجم).

باستخدام قيمة التسوية وزاوية الخط المستقيم الذي يشير إلى المقياس، يتم حساب موقع الرسم للنقطة على الخط. حتى تكون قيم القياسات مرئية على المخطط، يتم وصل قيم القياسات المتجاورة مشكّلة بذلك مضلّعاً، كالمبيّن في الشكل 9-2.

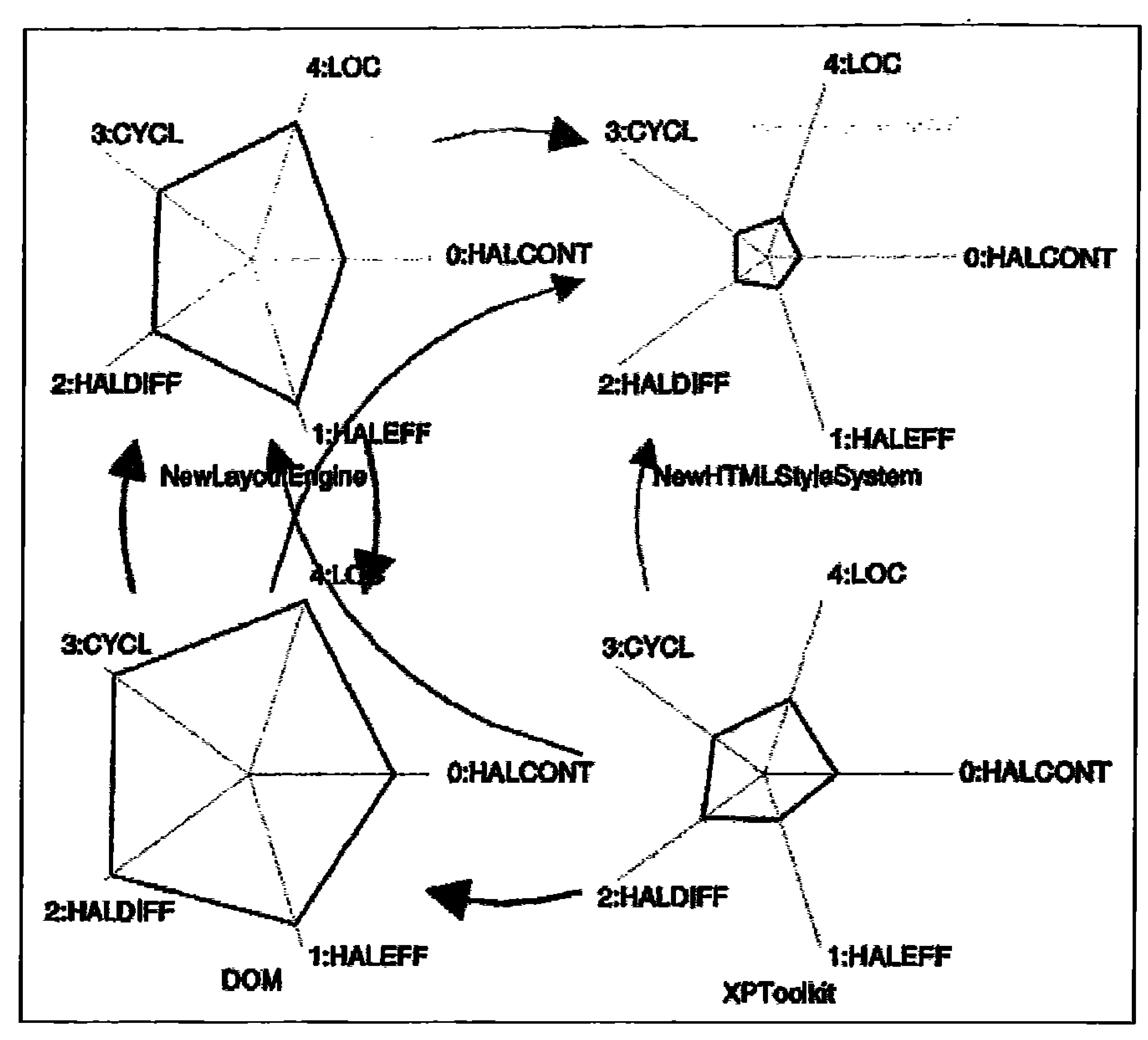


الشكل (9 ـ 2): مخطّط Kiviat للوحدة moduleA تمثل قراءات ستة مقاييس للشيفرة المصدرية M_1، M_2، ، ، ، ، M_6 لإصدار واحد

مخططات Kiviat البيانية هي عبارة عن نقاط في المخطط البياني الذي يمثل الوحدات. يمكن أن تتصل هذه النقاط بحواف تشير إلى علاقات الاستخدام التبعية بين الوحدات. ترسم الحواف على شكل أقواس تشير إلى اتجاه العلاقة.

يطبّق مبدأ العروض ذات المقاييس المتعددة على الحواف أيضاً عن طريق مطابقة عدد العلاقات التبعية المضمنة ليكون عدد استدعاءات العملية الثابتة نفسها بين وحدتين بالنسبة إلى عرض القوس.

هذا ويمكن تهيئة مجموعة المقاييس وترتيبها على المخطط. وينطبق الأمر ذاته على أنواع العلاقات التبعية والمقياس المطابق لعرض الأقواس، وهذا يتيح للمستخدم إنشاء عروض مختلفة عن عمليات التنفيذ ملقياً الضوء على جوانب معيّنة. على سبيل المثال، يوضح الشكل 9-3 عرضاً لأربعة محتويات ووحدات تصميم من إصدار Mozilla 1.7.



الشكل (9 ـ 3) : مخطط Kiviat بياني لمحتويات موزيلا (DOM) ووحدات التصاميم (NewLayoutEngine وNewHTMLStyleSystem وXPToolKit) مبيناً درجة تعقيد البرنامج وعدد أسطر الشيفرة البرمجية والمضمون القوي (الأقواس) للإصدار 1.7

المظاهر المصورة في هذا الرسم البياني تُعنى بتحديد الوحدات الكبيرة والمعقدة، إضافة إلى تلك التي تتضمن علاقات تبعية قوية في ما بينها. تعرض مخططات Kiviat البيانية الوحدات الأربع والأقواس بينها تمثل العلاقات التبعية المضمنة. في مخططات Kiviat يُمثّل حجم الوحدة بعدد أسطر الشيفرة البرمجية LOC وتمثل درجة تعقيد البرنامج تمثل بمقاييس هوليستيد (HALCONT, HALEFF, HALDIFF ومقياس ماكابي لدرجة التعقيد السيكلوماتي(*). عرض الأقواس يمثل قوة العلاقات التبعية المضمنة، في حين يتم حساب عدد هذه

(*) McCabe Cyclomatic Complexity (CCMPLX): مقياس أو مؤشر مكابي أو ما يعرف بدرجة التعقيد السيكلوماتي (Cyclomatic complexity) هو مقياس أو قيمة يمكن من خلالها قياس درجة تعقيد برنامج أو خوارزمية معينة (المترجم).

العلاقات التي تقطع حدود الوحدة. هذا ويشار إلى الوحدات الكبيرة والمعقدة بمضلعات كبيرة. وتمثل العلاقات المضمنة القوية بأقواس سميكة.

باستخدام طريقة الرسم هذه، يبيّن العرض بوضوح أن الوحدتين NewLayoutEngine وDOM (Document Object Model) (نموذج كائن الوثيقة) هما وحدتان كبيرتان ومركبتان. وبالمقارنة بهما، يتضح أن الوحدة NewHTMLStyleSystem هي وحدة صغيرة. كما يبيّن عرض الوحدة مدى قوة العلاقات بين الوحدات الأربعة. ما يثير الاهتمام هنا هو أن الوحدة DOM التي تنفذ وظائف المحتوى تتضمن عدداً كبيراً من الملفات من الوحدتين NewLayoutEngine وNewHTMLStyleSystem اللتين توفران وظائف تصميم صفحات الويب، وليس العكس، كما كنّا نتوقع.

إضافة إلى ما تقدم، ثمة علاقة مضمنة قوية ذات اتجاهين بين الوحدتين NewLayoutEngine وDOM. هذا ويجب مناقشة كافة جوانب القصور المحتملة مع المطوّرين لأنها تعيق تطوير وصيانة هذه الوحدات الثلاث.

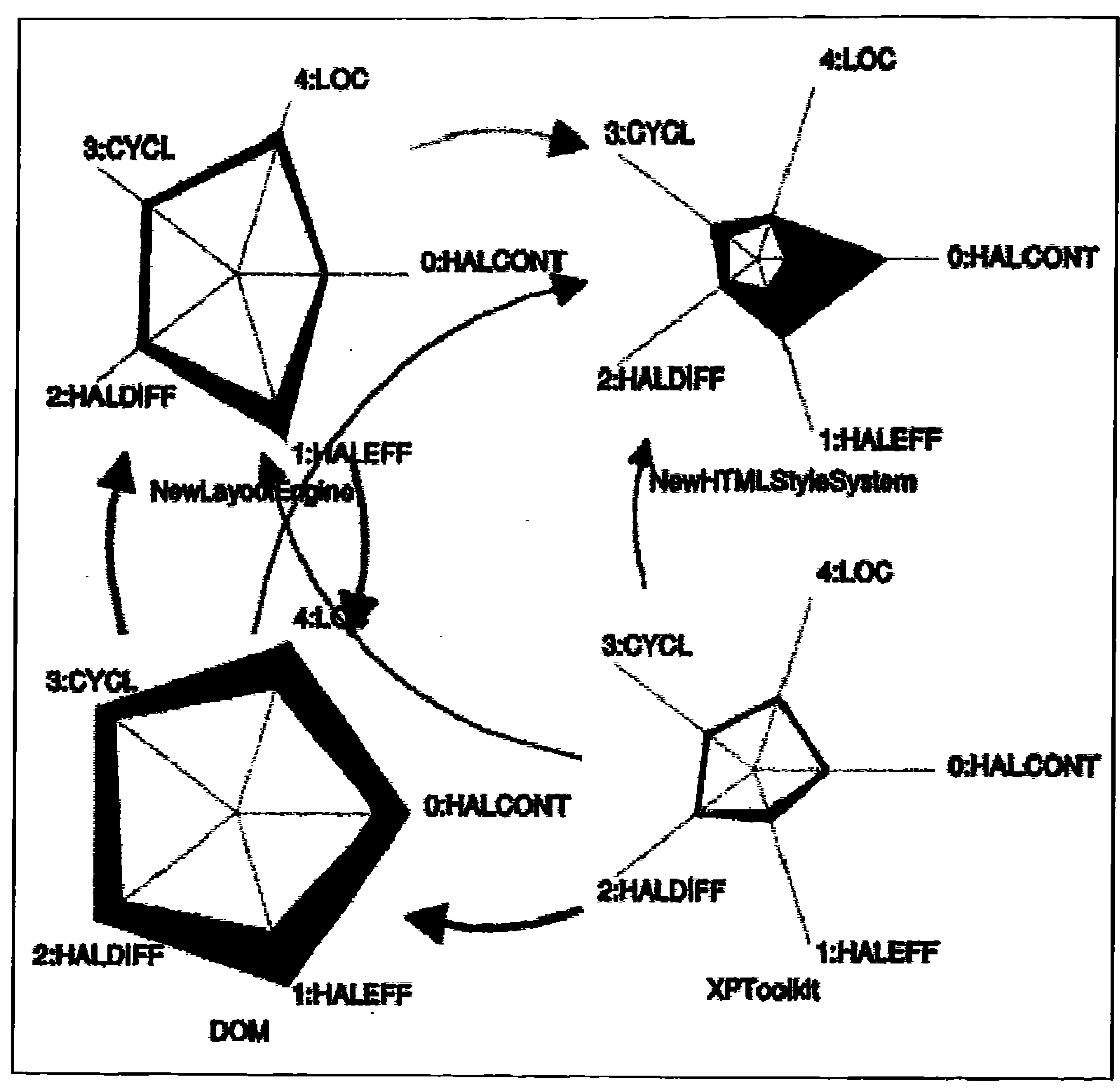
9 ـ 2 ـ 3 تصور قيم قياسات متعددة لإصدارات متعددة

عند تصور قيم القياسات لعدد من الإصدارات المتتالية يكون تركيزنا الأساسي منصباً على تحديد التغير بين قيم القياسات. إن الزيادة في القيم تشير إلى وجود إضافة وظائف، بينما يشير النقص في القيم إلى حذف من الوظائف. إضافة الوظائف هي إشارة عادة إلى تطور النظم البرمجية، فهي إذاً لا تشير إلى وجود مشكلة. على العكس من ذلك، فإن حذف وظائف يشير إلى تغير في التصميم. على سبيل المثال، نقل العمليات إلى نوع (Class) مختلف لحل العلاقة التبعية ذات الاتجاهين أو حذف عمليات بسبب حذف شيفرة برمجية لم تعد مستعملة.

لإلقاء الضوء على التغيير في قيم القياسات، نستخدم مخططات Kiviat البيانية، كما ورد سابقاً. يتم الحصول على القيم n لكل مقياس من الإصدارات المتعددة، ويتم رسمها على نفس المحور. يتم وصل قيم القياسات المتجاورة بخط لتشكّل مضلعاً لكلّ إصدار. يتم ملء المنطقة الناتجة بين مضلعين لإصدارين متتاليين بألوان مختلفة. يشير كل لون للتغيرات بين قيم قياسات إصدارين. كلما كان حجم التغيير كبيراً زاد حجم المضلع.

يبيّن الشكل 9-4 وحدات موزيلا الأربع نفسها كما كانت سابقاً، لكن هذه المرة بوجود بيانات المقاييس لثلاثة إصدارات متتالية هي 0.92 و1.3a و1.7. تشير المضلعات ذات اللون الرمادي الفاتح إلى التغيرات بين الإصدارين 1.3a و1.7.

يبيّن العرض تغيرات قوية في الوحدتين DOM وNewHTMLStyleSystem. في الوحدة الثانية، قلت قيم قياسات هولستيد HALCONT وHALDIFF بين الإصدار السابق والإصدار الأخير بشكل ملحوظ، على الرغم من أن الحجم (عدد أسطر الشيفرة البرمجية) لم يتغير كثيراً. من الواضح أن الشيفرة المصدرية لهاتين الوحدتين قد أعيد تصميمها.



الشكل (9 ـ 4): مخطط Kiviat لمحتويات Mozilla ونماذج العرض التي تبينّ درجة تعقد البرنامج ومقاييس عدد أسطر الشيفرة البرمجية وكثافة ثلاثة إصدارات فرعية متعاقبة 0.92 و1.3a و1.7.

زادت قيم القياسات لوحدة DOM في البداية، ثم قلّت في الإصدار الأخير مرة أخرى. أولاً أضيفت الوظائف إلى الوحدة التي أعيد تصميمها في أثناء تنفيذ الإصدار الأخير. بمقارنة هاتين الوحدتين، تشير قيم القياسات للوحدات الأخرى تغيراً طفيفاً في الحجم ودرجة تعقد البرنامج؛ ما يعني ثباتها. بناءً على افتراض أن الوحدات التي تغيّرت في الإصدار السابق ستتغير في الإصدارات المستقبلية فإن الوحدتين DOM وNewHTMLStyleSystem مرشحتان للتغير ما يعني ضرورة الاعتناء بهما.

يبيّن الشكل 9-4 مخطط Kiviat البياني لأربعة من محتويات موزيلا ووحدات التصميم مبيّناً درجة تعقيد البرنامج وعدد أسطر الشيفرة البرمجية والمضمون القوي (الأقواس) لثلاثة إصدارات متتالية هي 0.92 و1.3a و1.7.

9 ـ 3 عرض تطوّر ميزات النظام البرمجي

الميزات هي طريقة لعرض النظم البرمجية من وجهة نظر المستخدم. فكما يشير كانغ Kang وآخرون[12]، الميزة هي مظهر بارز أو مميز أو جودة أو سمة من سمات النظام أو النظم البرمجية. بما أن الميزات تستخدم في التواصل بين المستخدمين والمطوّرين فمن الأهمية بمكان معرفة الميزات التي تتأثر بالتعديلات الوظيفية في النظام. تركز تقنية تصور الميزة المقدمة في هذا القسم على إنشاء عروض ذات مستوى متقدم للميزات وتطبيقها في الشيفرة البرمجية المصدرية، إضافة إلى تبعياتها حسب تعديلات وأخطاء النظام. أما الأفكار الرئيسة ومفاهيم عروض تطور الميزة فقد طُوّرت في أبحاث فيشر Fischer وGall[15].

9 ـ 3 ـ 1 بيانات الميزات والتعديلات وأخطاء النظام

تنتقى بيانات الإدخال للتصور من قاعدة بيانات الإصدارات السابقة. تحصل تقارير التعديلات من نظم الإصدارات كـ CVS. فهي تتضمن معلومات عن من أجرى التعديل، وأي ملف مصدري معدل، ومتى تمّ ذلك. أما تقارير أخطاء النظام فيتم الحصول عليها من نظم تتبع الأخطاء كنظام Bugzilla. فهي من بين التقارير الأخرى تتضمن معلومات عن تاريخ الإبلاغ عن الخطأ والمكوّنات المتأثرة به ووصف مختصر للمشكلة وأولويتها ودرجة خطورتها وملاحظات عن الإصدارات الصغيرة التي ستتضمن حلولاً لها.

أما في ما يتعلق باستخلاص بيانات الميزة، فنحن نستخدم تقنية استطلاع

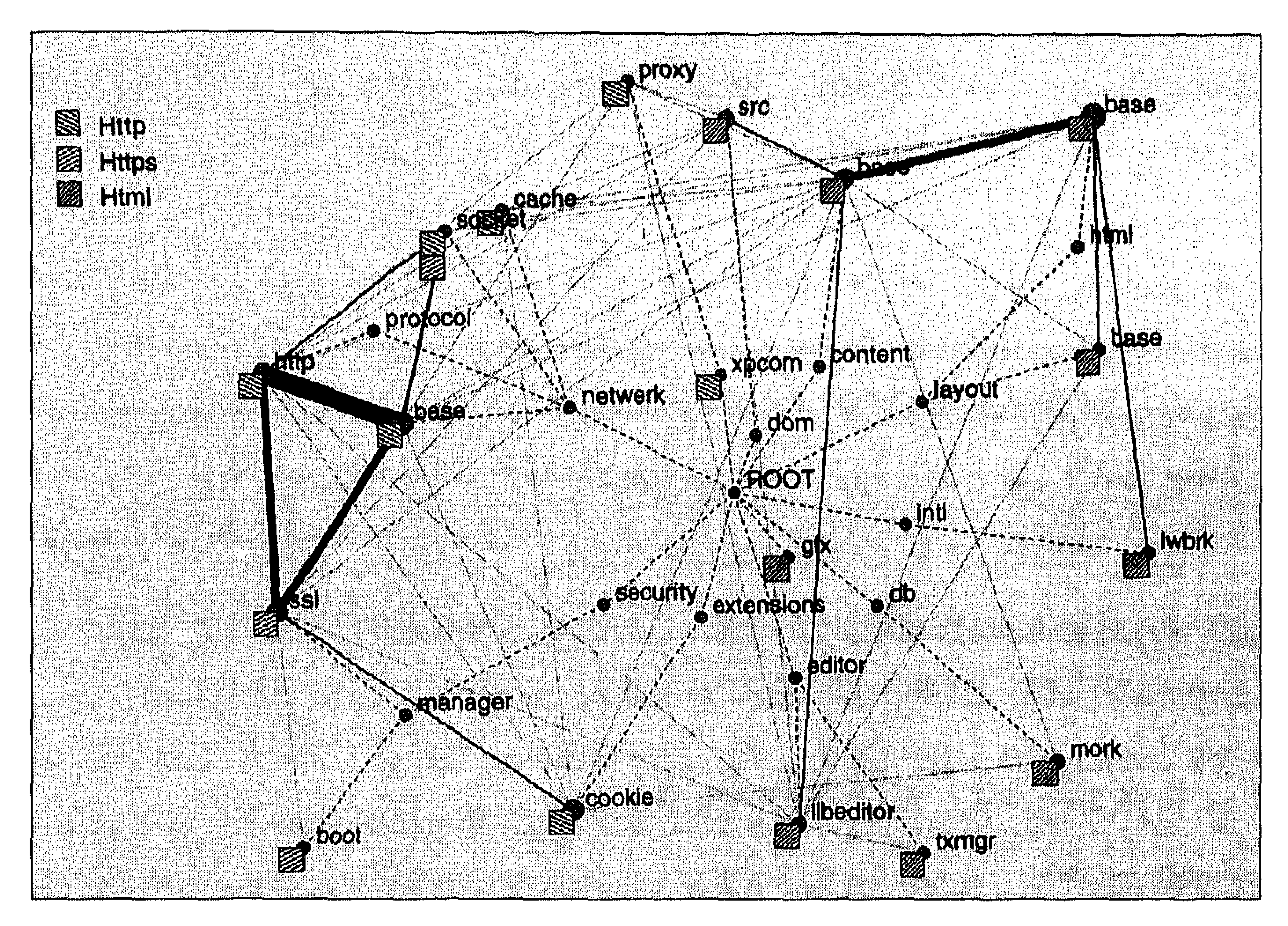
البرمجية التي قدمها كلٌ من ويلد Wilde وسكالي Scully[25]. ننفذ سيناريوهات استخدام مختلفة كتحميل موقع إلكتروني باستخدام موزيلا ونتتبع أثر التنفيذ باستخدام أداة إعداد ملفات التعريف (مثلاً، GNU gprof). يتضمن تتبع أثر التنفيذ تسلسل الوظائف (أي رسم بياني للاستدعاء) المنفذة في كلّ سيناريو لكلّ مستخدم. ومن ثم تلخص تتبعات تنفيذ السيناريوهات المختلفة على مستوى الملفات المصدرية ويتم تعيينها للمفاهيم التي تمثل الميزات. ثم يتم تخزين مجموعة الوظائف والملفات المصدرية التي تنفذ ميزة معيّنة في قاعدة بيانات الإصدارات السابقة.

يتم ربط تقارير التعديلات وبيانات الميزة بواسطة ملفات مصدرية. يتم توفير ربط بيانات ملف التعديلات وملف الأخطاء إما بواسطة نظام الإصدارات أو نظام تتبع الأخطاء أو قد يتم إعادة بنائه. بالنسبة إلى نظام الإصدارات CVS ونظام تتبع الأخطاء Bugzilla فهي الحالة الأخيرة. يتم إعادة بناء الروابط عن طريق الاستعلام عن وصف التعديل (أي ملاحظات الاعتماد[*] في CVS) للإشارة إلى تقارير الأخطاء (e.g., bug #2345). عندما يتم العثور على هذه الإشارة يتم تخزين رابط بين التعديل وتقرير الخطأ المرتبط به في قاعدة بيانات الإصدارات السابقة. لمزيد من التفاصيل حول استخلاص بيانات الميزة وخوارزميات تكامل البيانات، يرجى مطالعة أعمال Fischer وآخرين[6, 5].

9 ـ 3 ـ 2 عرض المشروع ـ إبراز تقارير الأخطاء على هيكلية الفهرس

يصور عرض المشروع تنفيذ الميزات بواسطة الملفات المصدرية وتقارنها من خلال تقارير الأخطاء باستخدام الرسم البياني. تمثل النقاط فهارس الشيفرة البرمجية المصدرية، تشير الحواف المتقطعة ذات اللون الرمادي إلى تسلسل الفهرس الهرمي (شجرة المشروع). تمثل الميزات بمستطيلات مرفقة بنقاط الفهرس وتحتوي على الملفات المصدرية التي تنفذها. أما الأقواس الرمادية المتصلة في الرسم البياني فتشير إلى التقارن بين الميزات من خلال تقارير الأخطاء. يوضح الشكل 9-5 مثالاً على مثل هذا العرض لثلاث من ميزات موزيللا.

(*) الاعتماد (Commit) في هذا السياق يعني تحويل مجموعة من التغيرات المحتملة إلى تغيرات دائمة (المترجم).



الشكل (9 ـ 5): علاقات الربط والتطبيق لبنية موزيللا، Mozilla Http، Https، Html من خلال تقارير خطأ[5]

شكلياً، تكون النقطتان vi وvj في الرسم البياني متصلتين إذا كان فهرسان اثنان يحتويان ملفات مصدرية تتشارك في تقرير خطأ (أي ملفات تم تعديلها لإصلاح الخطأ). يتم احتساب أوزان الحواف بين النقاط باستخدام المعادلة الآتية، حيث إن n تحدد عدد الارتباطات الحالية بين النقطتين وn_{max} هو العدد الأقصى للارتباطات (العامة) بين أي نقطتين في الرسم البياني:

$$\text{weight}\ (v_i, v_j) = (-1)\left(\frac{n}{n_{\max}}\right)^k + o$$

عندما تكون o=1، فإن جميع الأوزان تُعيّن للمدى [0. . . 1]، حيث 0 يعني أقرب مسافة (k تحكم المسافة الناتجة بين النقاط وثيقة الصلة). في الرسم البياني الناتج، أما الخطوط السميكة الأغمق لوناً فتعني أن عدد تقارير الأخطاء التابعة المشتركة بين نقطتين كبير. نستخدم الأداة Xgvis[22] لتصميم الرسم البياني (وهي تطبق خوارزمية التحجيم متعدد الأبعاد).

من الخطوات الحاسمة في عملية إنشاء الرسم البياني خطوة اختيار معايير

الأوزان، ذلك أن لها تأثيراً مباشراً في التصميم النهائي. استخدمنا النسبة 20:1 لحواف الشجرة وحواف تقرير الأخطاء. هذا المخطط يعطي تركيزاً على هيكلية الفهرس أكثر من الارتباطات التي يعرفها تقرير الخطأ. في الخطوة الأولى من عملية إنشاء البيانات، يتم تعيين كوائن شجرة الفهرس للنقاط الخاصة بها على الرسم البياني. يحدد أدنى حجم لفرع في الشجرة (minchild) النقاط الموسعة (أي التي تبدو فروعها) أو تلك التي سيتم طيّها (أي إخفاء فروعها). الطي يعني أن الكوائن التابعة للشرعة الفرعية ستنقل إلى المستوى الأعلى الأقرب إلى أن يتحقق معيار الحجم، لكن الانتقال لا يتجاوز أول مستوى تحت النقطة الجذرية (ROOT node). الفهارس غير المؤشرة يتم إخراجها. في الخطوة الثانية، يكون من الممكن نقل نقاط ذات مؤشر أقل إلى مستوى أعلى للحصول على تمثيل أكثر تراصاً. أما التأثير في الرسم البياني فهو أن الأطراف غير المؤشرة يتم منعها، على الرغم من أنها تحوي ما يكفي من الكوائن لتحقيق معيار الفرع الأدنى minchild. وهذا يقلّل كمية المعلومات التي سيتم تصورها أكثر ما يحسن من فهم الرسوم البيانية الناتجة.

في الأقسام الفرعية الآتية نبيّن مثالين لعروض المشروع، كما تم إنشاؤها في دراسة حالة خاصة بمشروع Mozilla ذي المصدر المفتوح.

9 ـ 3 ـ 3 تقارن تقرير الخطأ بين خصائص: Mozilla Http وHttps وHtml

الخصائص الثلاث Http وHttps وHtml موضحة في الشكل 9-5. اخترنا جميع تقارير الأخطاء كبيانات إدخال لهذه الخصائص باستثناء التقارير المصنفة كـ «تحسينات» منذ بداية المشروع حتى تاريخ التوقف في 10 كانون الأول/ ديسمبر 2002. لأغراض التصور، قمنا بتهيئة الخوارزمية باعتبار أن الفرع الأدنى minchild = 1 (أي عدد الملفات في الفرع الأدنى) وباعتبار أن التراص = 1 (أي عدد تقارير الأخطاء المؤشر عليها من قبل النقطة).

بالنسبة إلى عملية التسوية، قمنا بوزن أطراف شجرة المشروع بـ 20، حيث كان كل طرف معرف بواسطة تقرير خطأ واحد قد وزن بـ 1. استخدمت العوامل التالية لمعادلة الوزن للتشديد على الانتشار بين النقاط لأغراض التصور: k = 0.2 وo = 0.2.

كمية تقارير الأخطاء الإجمالية التي يتم الكشف عنها عند نقطة واحدة يشار إليها من خلال قطر النقطة، أما الميزات المستضافة من قبل النقطة وتكون

مرفقة على هيئة صناديق. من السهل إدراك وضع النقاط التي تنتمي إلى خاصية Html في الطرف الأيمن في الشكل 9-5 وخاصيتي Http وHttps في الطرف الأيسر. النقاط network/base وnetwork/protocol/http وsecurity/manager/ssl هي نقاط مهمة، وذلك أنها متقارنة من خلال 90 تقريراً من تقارير أخطاء الطرف base-http، و40 تقريراً من تقارير أخطاء الطرفين الآخرين. وهذا يشير إلى وجود درجة عالية من التقارن بين الخاصيتين Http وHttps.

من الجوانب الأخرى المهمة انتشار خاصية Html خلال 10 نقاط مختلفة. قد يصعب تتبع التعديلات لوجود العديد من الملفات في الفهارس المختلفة التي تشترك في ميزة واحدة. أما النقطتان content/base وlayout/html/base فهما جديرتان بالاهتمام، ذلك أنهما متقارنتان من خلال 35 من تقارير الأخطاء، على الرغم من وجود أربعة ملفات في فهارس النقطة الأولى، وثلاثة ملفات في فهارس النقطة الثانية فقط.

نتيجة لذلك، يبيّن الشكل 9-5 تبعيات تغيير قوية للخصائص المعنية خلال الفهارس المختلفة، ويشير أيضاً إلى تدهور الهيكلية نتيجة تطور البيانات:

1. قد تشير الخصائص المنتشرة في شجرة المشروع إلى تطور كمية كبيرة من الشيفرة البرمجية الأساسية، وبالتالي فإن تأثير التغيير يكون كبيراً.

2. قد تشير التبعيات بين فروع شجرة المشروع إلى وجود تبعيات أخرى كالاستدعاء أو الوصولية أو التوارث أو الارتباط.

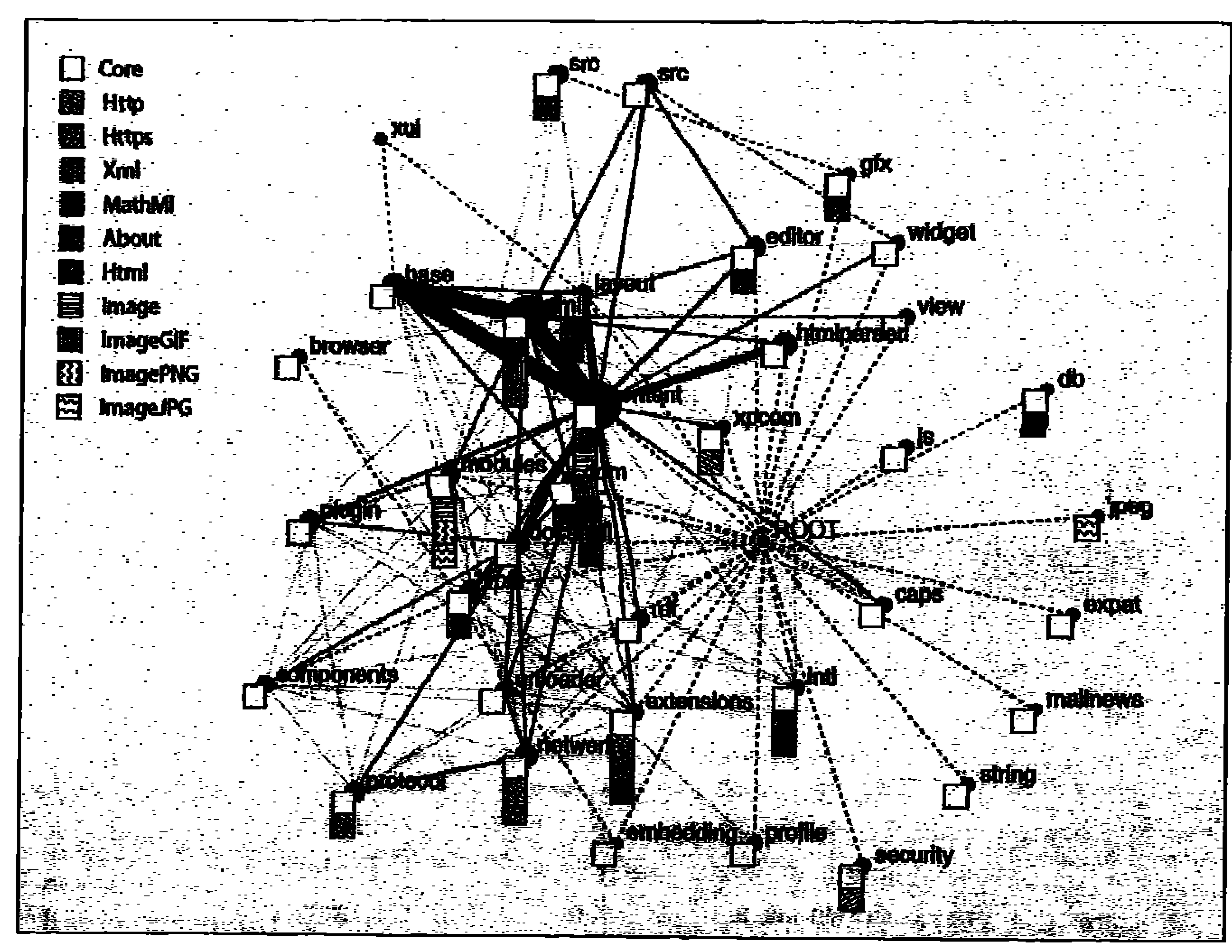
3. الأخطاء التي يتم الإبلاغ عنها بشكل متكرر المتعلقة بالشيفرة البرمجية المصدرية لمواقع الخصائص قد تكون مؤشراً على مشكلات في التنفيذ أو التصميم.

9 ـ 3 ـ 4 تقارير الأخطاء المتقارنة بين خصائص وأساسات Mozilla

يبيّن الشكل 9 ـ 6 الخاصية الأساسية (ملفات مصدرية غامضة قابلة للتخصيص) وجميع الخصائص الأخرى المتحقق منها على مستوى تفصيلي (الأطراف التي تمثل أقل من خمسة مؤشرات يتم حذفها). للحصول على هذه التهيئة قمنا باختيار جميع التقارير المصنفة «رئيسة» أو «حرجة». كما قمنا بتحديد أدنى حجم للشجرة الفرعية بـ 250 (كائناً) (أدنى أطراف) وأدنى عدد لمؤشرات الفروع بـ 50 (متراص). نتج من ذلك مخطط بياني ذو 37 نقطة و 315 طرفاً ناجمة

عن تقارير الأخطاء. بتغير قيم الأطراف الدنيا والتراص، يصبح من الممكن إنشاء مخطط بياني تفصيلي تعسفياً للمشروع كاملاً. بما إن موزيللا تتضمن أكثر من 2500 فهرس فرعي، يكون من الصعب الحصول على مخطط بياني يمثل شجرة المشروع كاملة نظراً إلى إمكانيات التوضيح المحدودة. من البدهي أن ترتبط معظم النظم الفرعية في موزيلا في التصوّر وقد دعمت نتائجنا هذا.

النقاط الأعلى كثافة في تقارير الأخطاء المصنفة على أنها الرئيسة والحاسمة هي content (608 مؤشر) وlayout/html (444) وlayout/xul (223) وlayout (212). من الجوانب الأخرى المهمة انتشار الأطراف. لذا، فإن إجمالي عدد الارتباطات الموصوفة بين النقاط يكون 343. فإذا اخترنا الأطراف التي تمثل 10 مؤشرات فحسب فإننا نستطيع إيجاد الرتب الآتية: content بـ 19 طرفاً وlayout/html بـ 10 أطراف وdocsell بـ 9 أطراف وdom بـ 5 أطراف وlayout/xul/base وnetwork وuriloader بـ 4 أطراف لكلّ منها. في المجموع، تتشارك 23 نقطة الأطراف مع نقاط أخرى بعشرة مؤشرات على الأقل، لكن 19 نقطة منها ترتبط بالنقطة content. وهذا يؤكد الموقع الاستثنائي للمحتوى الذي يشار إليه بست خصائص مختلفة موجودة هناك.



الشكل (9 ـ 6): العلاقة الوظيفية بين بنى Mozella، وبنى Core

نتيجة لذلك، تتيح الصور كالشكل 9 ـ 6 للمحللين رسم الاستنتاجات الآتية:

1. تظهر النقاط ذات التغيير المتكرر بحجم أكبر من النقاط الأخرى ويمكن رصدها بسهولة.

2. الأجزاء غير المستقرة من النظام كالمحتوى يكون موقعها بالقرب من مركز المخطط البياني وتكون شديدة التقارن بالنقاط الأخرى.

3. الميزات ذات الشيفرة البرمجية المشتركة ترفق بنقاط محددة وتكون قريبة من بعضها البعض (مثال ذلك MathMI وXML).

4. مجموعات ميزات محددة (مثال، ميزة الصور) المنتشرة عبر نقاط عديدة يمكن رصدها بسهولة (مثال، النقاط jpeg وmodules وlayout/html وcontent).

كنتيجة لذلك، المواقع ذات التغيير المكثف وانتشار الميزات تشيران إلى أجزاء البرمجية التي يجب أن تراعى لإجراء مزيد من التحقق من حيث الحد من التعقيد الكبير أو تدهور الهيكلية.

9 ـ 4 عرض إسهامات المطوّرين

في هذا القسم، نركز على تصور الجهود التي بذلها المطوّرون لتصحيح الأخطاء وتطوير النظم البرمجية. إن الهدف الأساسي لعملية التصور هذه هو توفير رؤى عن عدد المرات التي تغيرت فيها وحدة الشيفرة البرمجية المصدرية ومن أجرى تلك التغيرات. تبيّن العروض الناتجة أنماطاً تمكننا من التفكير في سلوك التغير لوحدات الشيفرة البرمجية المصدرية، كما لو أن هناك مطوّراً مركزياً أو عدة مطوّرين يجرون هذه التغيرات. الأفكار والمفاهيم الأساسية لعروض الكسور (Fractal Views) طورت من خلال أعمال دامبروس D'Ambros وآخرين[2].

9 ـ 4 ـ 1 بيانات التعديل

كما في المنهجية السابقة، نحصل على تقارير التعديلات من قاعدة بيانات الإصدارات السابقة RHDB ونقوم بحساب عدد الاعتمادات لكل مؤلف وملف مصدري. على سبيل المثال، يبيّن الجدول 9-1 المطوّرين الذين عملوا على الملف المصدري nsTextHelper.cpp الخاص بموزيللا وعدد الاعتمادات التي نفذها كلّ منها.

الجدول (9 ـ 1)

المؤلفون وعدد الاعتمادات التي نفذوها على الملف المصدري nsTextHelper.cpp الخاص بموزيللا[2]

المؤلف	عدد الاعتمادات
warren@netscape.com	5
gerv@gerv.net	2
dcone@netscape.com	2
dbaron@fas.harvard.edu	1
cltbld@netscape.com	1
Pierre@netscape.com	1
dmose@mozilla.org	1
juggernaut@netscape.com	1
8 مطوّرون	**14 اعتماد**

9 ـ 4 ـ 2 العروض الكسيرية

تصور العروض الكسيرية (Views Fractal)(*) جهود التطوير التي بذلها كل مؤلف في العمل على وحدات الشيفرة البرمجية المصدرية. لهذا العرض، تشير وحدات الشيفرة المصدرية إلى الملفات المصدرية. تمثل كل وحدة بشكل كسيري (Fractal) يرسم على شكل مستطيل يتكون من مجموعة من المستطيلات المملوءة ذات أحجام وألوان مختلفة. يعين كل مستطيل (وبالتالي كل لون) لمؤلف يعمل على الملف. وباستخدام مبدأ مطابقة القياس، تتناسب مساحة المستطيل مع نسبة جهود التطوير المبذولة من قبل المؤلف من إجمالي الجهد. وبالتالي فإنه كلما زاد الجهد الذي يبذله المطوّر في العمل على وحدة

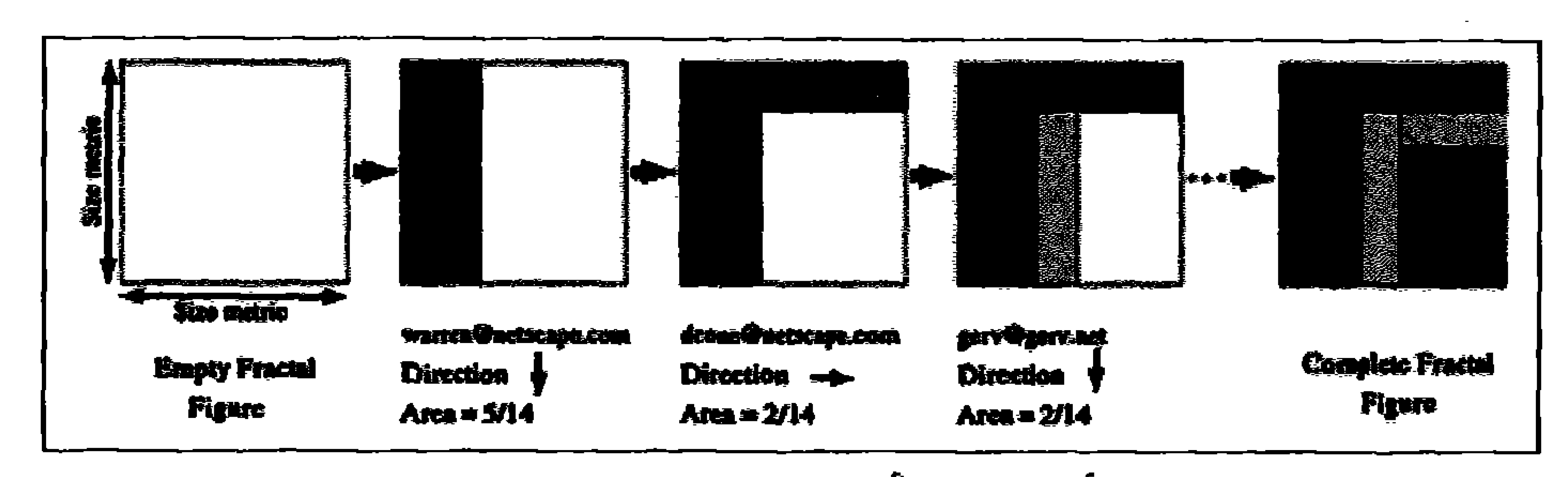
(*) يمكن تعريف الكسيرية على أنها كائن هندسي خشن غير منتظم على كافة المستويات، ويمكن تمثيلها بعملية كسر شيء ما إلى أجزاء أصغر، لكن هذه الأجزاء تشابه الجسم الأصلي. تحمل الكسيرية في طياتها ملامح مفهوم اللانهاية، وتتميز بخاصية التشابه الذاتي، أي إن مكوّناتها مشابهة للكسيرية الأم مهما كانت درجة التكبير. غالباً ما يتم تشكيل الأجسام الكسيرية عن طريق عمليات أو خوارزميات متكررة: مثل العمليات التراجعية (Recursive) أو التكرارية (Iterative) (المترجم).

معيّنة، كان المستطيل الذي يمثله أكبر. يعطى الجهد كعدد الاعتمادات لكنه ليس محصوراً بالعدد. يمكن أيضاً استخدام مقاييس أخرى كعدد أسطر الشيفرة البرمجية المضافة أو المحذوفة.

المثال التالي المبين في الشكل 9-7 والخاص بالملف nsTextHelper.cpp يشرح عملية بناء عروض الكسور. أخذت البيانات من الجدول 9-1.

المستطيل الأول ذو اللون الرمادي الغامق في الطرف الأيسر يمثل المؤلف الذي أجرى أكبر عدد من الاعتمادات وهو تحديداً warrene@netscape.com. مساحة المستطيل هي 14/5 من إجمالي المساحة ذلك أن عدد الاعتمادات التي نفذها هذا المؤلف هو 5 بينما العدد الإجمالي هو 14. رسمت المستطيلات الأخرى بالطريقة نفسها مع تغيير اتجاه الجانب الأطول كل مرة.

يتغير اتجاه الرسم لتحسين التدرج في تقنية التصور. حتى عندما يكون هناك مئات المؤلفين ستبيّن الأشكال الكسيرية أن التطوير مجزأ بصورة كبيرة.



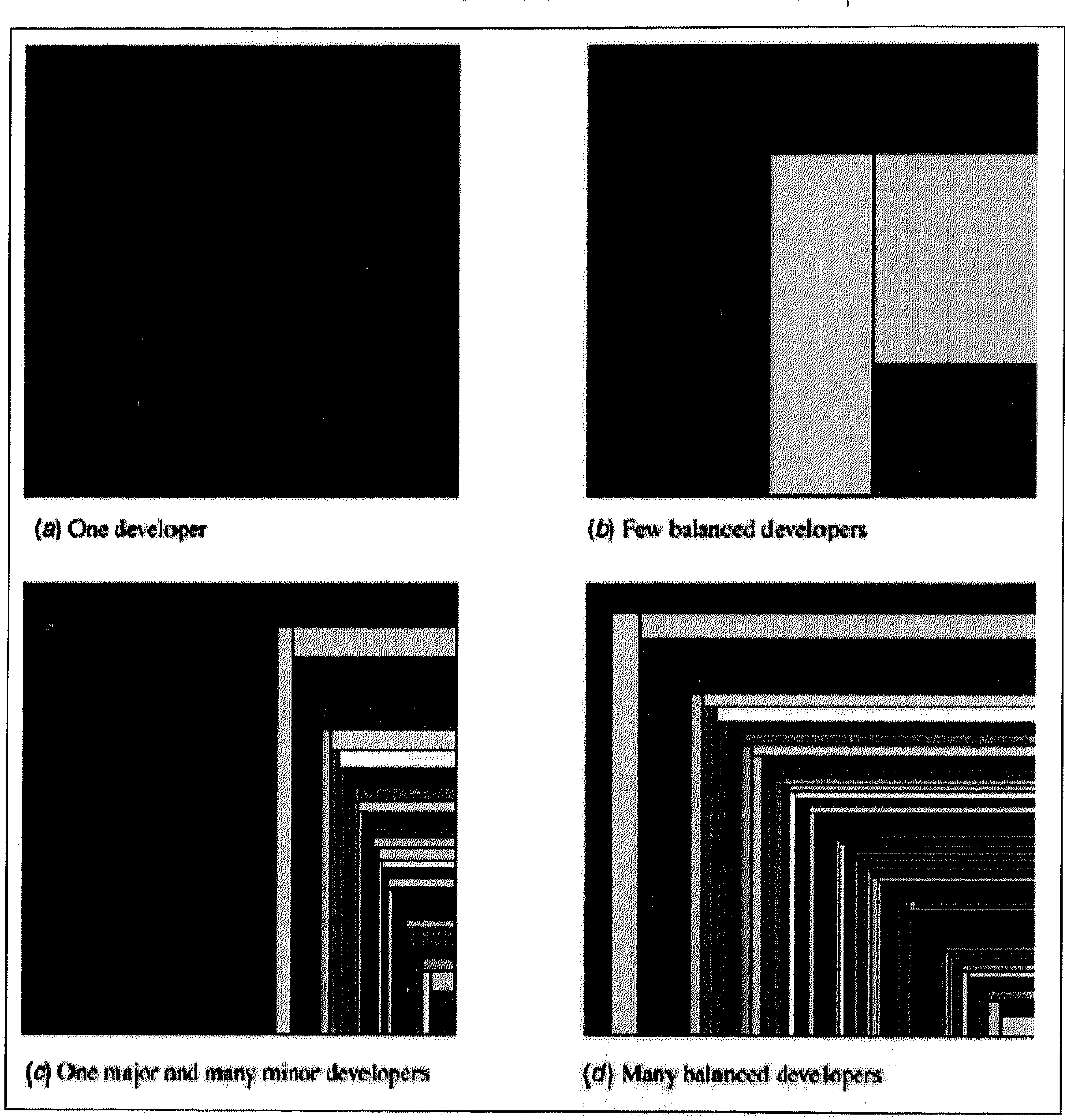
الشكل (9 ـ 7): مبدأ إنشاء الأشكال الكسيرية للملف Mozilla المصدري nsTextHelper

9 ـ 4 ـ 3 تصنيف الملفات المصدرية مع العروض الكسيرية

تستخدم العروض الكسيرية للتحقق من جهود تطوير الكوائن من وجهة نظر المؤلفين. يمكننا مقارنة الكوائن بالنظر إلى أشكالها وتصنيفها حسب أنماط التطوير. بإمكاننا التمييز بين أربعة أنماط تطوير رئيسة، كما هو مبين في الشكل 9-8: بوجود مطوّر واحد فقط (أ)، بوجود بعض المطوّرين والجهد متوازن (ب)، بوجود عدد كبير من المطوّرين لكن الجهد غير متوازن ـ أحدهم يقوم بنصف العمل والباقون يقومون بأداء النصف الآخر (ج)، وبوجود

العديد من المطوّرين الذين ينجزون مقدار العمل نفسه تقريباً (د).

يمكن أن تستخدم العروض الكسيرية أيضاً عندما تكون وحدات الشيفرة البرمجية ذات مستوى متقدم كفهارس الشيفرة البرمجية المصدرية. لهذا الغرض، يتم تجميع مقاييس الكوائن المحتواة (أي الملفات المصدرية). على سبيل المثال، عدد إصدارات الفهرس هو مجموع أعداد إصدارات الملفات المصدرية المحتواة. إضافة إلى ذلك، توسع التعبير عن العروض الكسيرية بواسطة مقاييس المطابقة لتشمل حجم المستطيلات. وهذا يسمح لنا تصنيف هذه الكوائن ذات المستوى المتقدم من حيث جهود التطوير وتوزيعه.



الشكل (9 ـ 8): أنماط التطوير بناء على أشكال غيشتالت (Gestalt) والأشكال الكسيرية[2]

على سبيل المثال، يبيّن الشكل 9-9 التسلسل الهرمي للفهرس webshell التابع لموزيللا. تمثل الأشكال الكسيرية الفهارس التي تتضمن ملفاً مصدرياً واحداً على الأقل، بينما ترتبط الأشكال رمادية اللون مع الفهارس الحاوية ـ بمعنى أنها الفهارس التي تحتوي على فهارس فرعية فقط. نستخدم عدد الملفات كمقياس للحجم. الشكل المرقم بـ 1 يمثل أكبر فهرس ـ أي هو الفهرس الذي يحتوي على أكبر عدد من الملفات المصدرية. فهو يعرض نمط التطوير «العديد من التطوير ـ جهد متوازن».

إن مجموعة الأشكال المرقمة ب 2 تُميّز بما يأتي:

(1) الأشكال التي تنتمي إلى الهيكل الهرمي نفسه.

(2) تعرض مطوّراً واحداً أو مطوّراً رئيساً واحداً.

(3) بعض الفهارس يتضمن عدداً كبيراً جداً من الملفات.

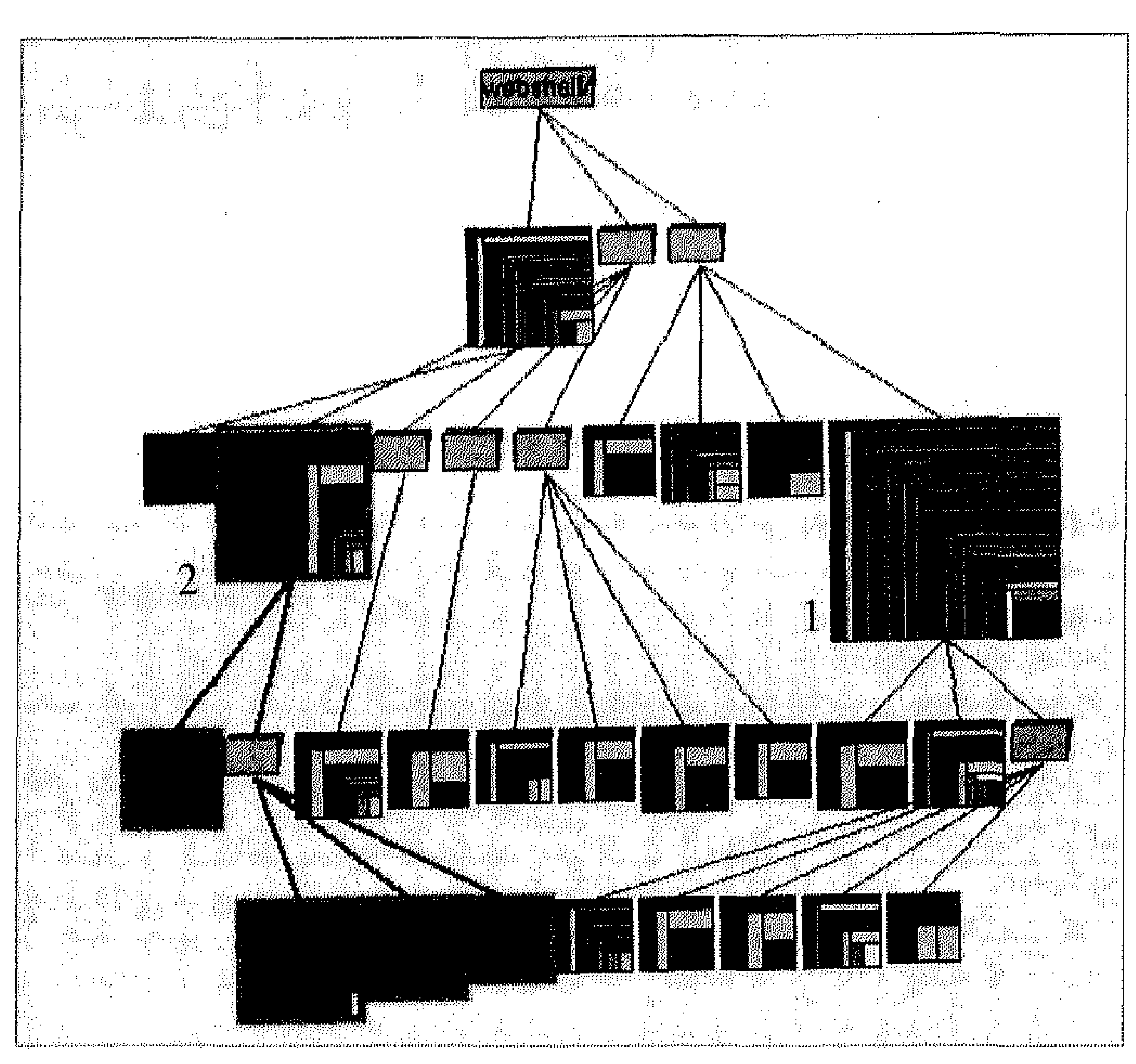
ثمة مثال آخر مأخوذ من دراسة حالة موزيللا معطى في المثال 9-10. العرض الكسيري يوجد في الفهرس editor/libeditor/html. الأشكال الكسيرية تمثل الملفات المصدرية (على الجانب الأيسر) والفهارس (الزاوية اليمنى العليا). نستخدم عدداً من الإصدارات لأحجام المستطيلات.

الأشكال المستطيلة التي تمثل الملفات المصدرية تشير إلى ملف واحد (المرقم بـ 1) وهو أكثر الملفات تعديلاً من قبل عدة مطوّرين. يشير نمط التطوير لهذا الملف إلى وجود عدد من المتطورين بجهد متوازن. أما المستطيلات المرقمة بـ 2 فهي أيضاً مهمة لأنها تظهر سلوك التغيير نفسه. من المرجح أن تكون متقارنة من حيث التغيير (أي إنه في الأغلب ما يتم تعديلها معاً) لأن:

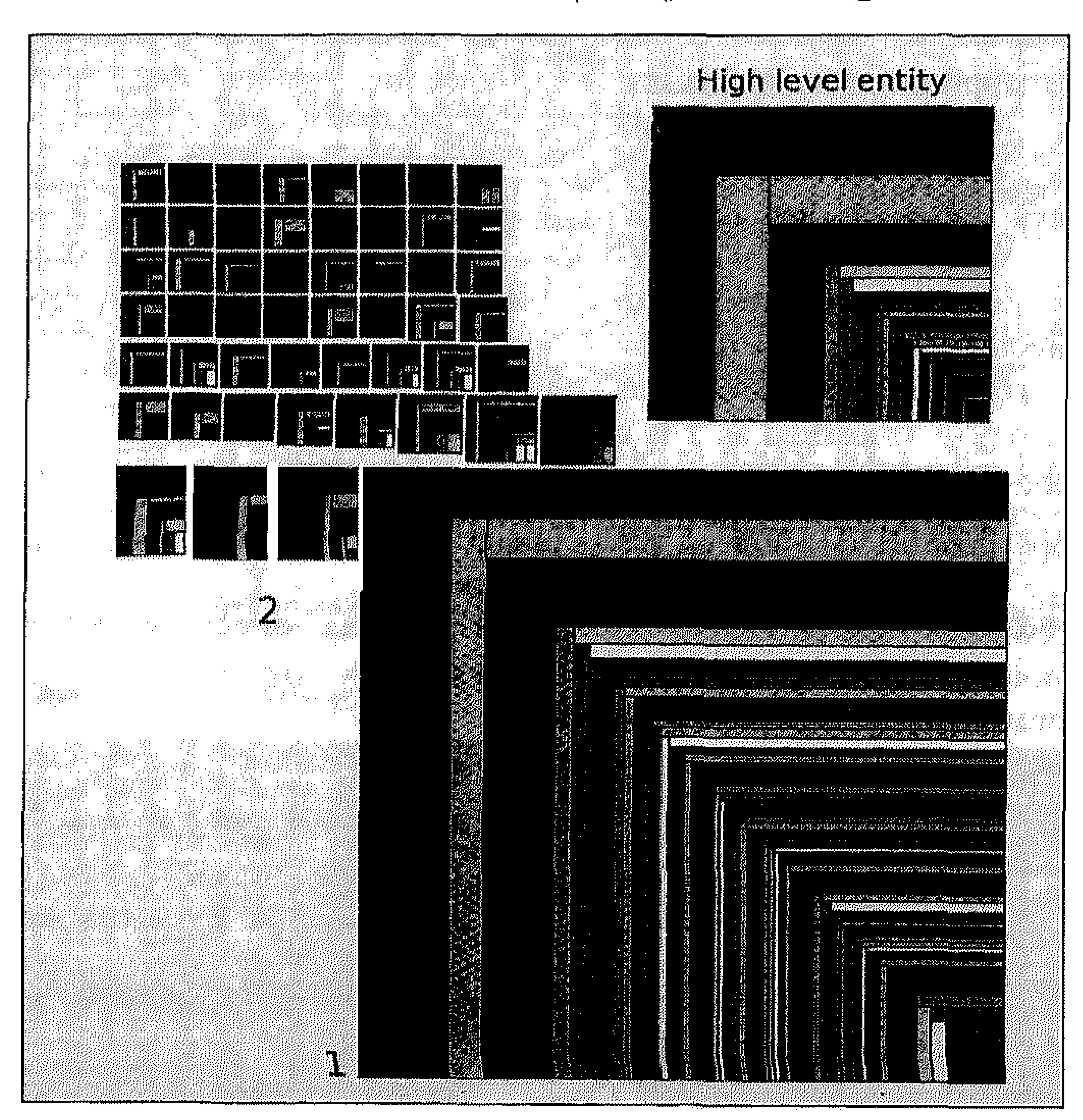
1) لها نمط البرمجة نفسه ومظهراً متشابهاً.

2) نمط البرمجة الخاص بها يختلف عن تمييز الفهرس.

يعكس الشكل الكسيري الذي يمثل الفهرس أنماط الملفات المصدرية المضمنة: هناك العديد من المطوّرين الذين يعملون بتوازن، لكن المساهمة الأعظم تأتي من مطوّرين اثنين وبعض المطوّرين الآخرين المخادعين.



الشكل (9 ـ 9): توزع المُظهر في نظام تراتبية webshell لموزيللا



الشكل (9 ـ 10): توزع المُظهر لنظام (editor/libeditor/html directory) في موزيللا[2]

9 ـ 5 عرض تقارن التغيير

يحدث تقارن التغيير عندما يُجرى تغير على وحدتين برمجيتين أو أكثر من قبل المؤلف نفسه وفي الوقت نفسه، تقريباً. في هذا القسم، نقدم تقنية EvoLens التصورية التي تركز على إلقاء الضوء على تقارن التغيير بين الملفات المصدرية والوحدات البرمجية. تشير الوحدات البرمجية إلى الفهارس في شجرة المشروع. ظهرت الأفكار والمفاهيم الرئيسة لعروض تقارن التغيير وطورت في أعمال راتزنغر Ratzinger وآخرين[21].

إن هدف عروض تقارن التغيير هو تحديد الملفات المصدرية والوحدات التي تتغير مع بعضها البعض في معظم الأحيان. يمكن أن تُستخدم هذه المعرفة، على سبيل المثال، كنقاط بداية لتحديد جوانب القصور في التصميم ولتقييم تأثير التغيير (مثلاً عند تغيير هذه الوحدة فإن وحدات أخرى يجب أن تتغير).

9 ـ 5 ـ 1 بيانات تقارن التغيير

يتم الحصول على بيانات عروض تقارن التغيير من قاعدة بيانات الإصدارات السابقة. فهي تضم معلومات فهرس الشيفرة البرمجية المصدرية وعدد الاعتمادات لكل ملف مصدري وعلاقات تقارن التغيير بين الملفات. حالياً، بنيت قاعدة البيانات من بيانات CVS. نظراً إلى أن CVS لا يدعم عمليات الاعتماد commit، يجب أن يعاد بناء علاقات تقارن التغيير بين الملفات المصدرية. نستخدم معلومات المؤلف والبيانات ومعلومات رسالة الاعتماد لإعادة بناء هذه العلاقات. بشكل أساسي، فإن تقارير التعديلات التي يجريها المؤلف نفسه في رسالة اعتماد واحدة واعتماد الوقت نفسه $t \pm e$ تُجمع في حركة واحدة. يتم تأسيس علاقات تقارن التغيير بين كل مجموعة من هذه المجموعات وتخزن في قاعدة بيانات الإصدارات السابقة.

9 ـ 5 ـ 2 عروض EvoLens

منهجية EvoLens هي تقنية تصور تفاعلية قائمة على المخططات الرسومية تمثل هيكلية الفهرس والملفات المصدرية كمخطط رسومي متداخل. يتم تمثيل الملفات المصدرية بشكل إهليلجي، ويتم إحاطة الفهارس بمستطيلات. أما تبعيات تقارن التغيير بين الملفات المصدرية فتمثّل على شكل خطوط مستقيمة بين العقد.

تتبع منهجية EvoLens مبدأ مطابقة المقاييس الموضح مسبقاً. بالنسبة إلى

الملفات المصدرية، يتم مطابقة معدل النمو في عدد أسطر الشيفرة البرمجية المضافة أو المحذوفة مع ألوان النقاط في الشجرة. أساسياً، اللون الخفيف يشير إلى نمو أدنى واللون الكثيف يشير إلى نمو أقصى. أما في ما يتعلق بمقارنة التغيير فيتم مطابقة عدد الاعتمادات المشتركة بين ملفين مع عرض الأقواس، فكلما كان عدد الاعتمادات المشتركة بين ملفين مصدريين أكبر، كان عرض القوس التي تصل بين نقطتين مرتبطتين أثخن. في ما يأتي، نوضح مفاهيم التصور المختلفة مع أمثلة مأخوذة من دراسة حالة خاصة بنظام أرشفة الصور.

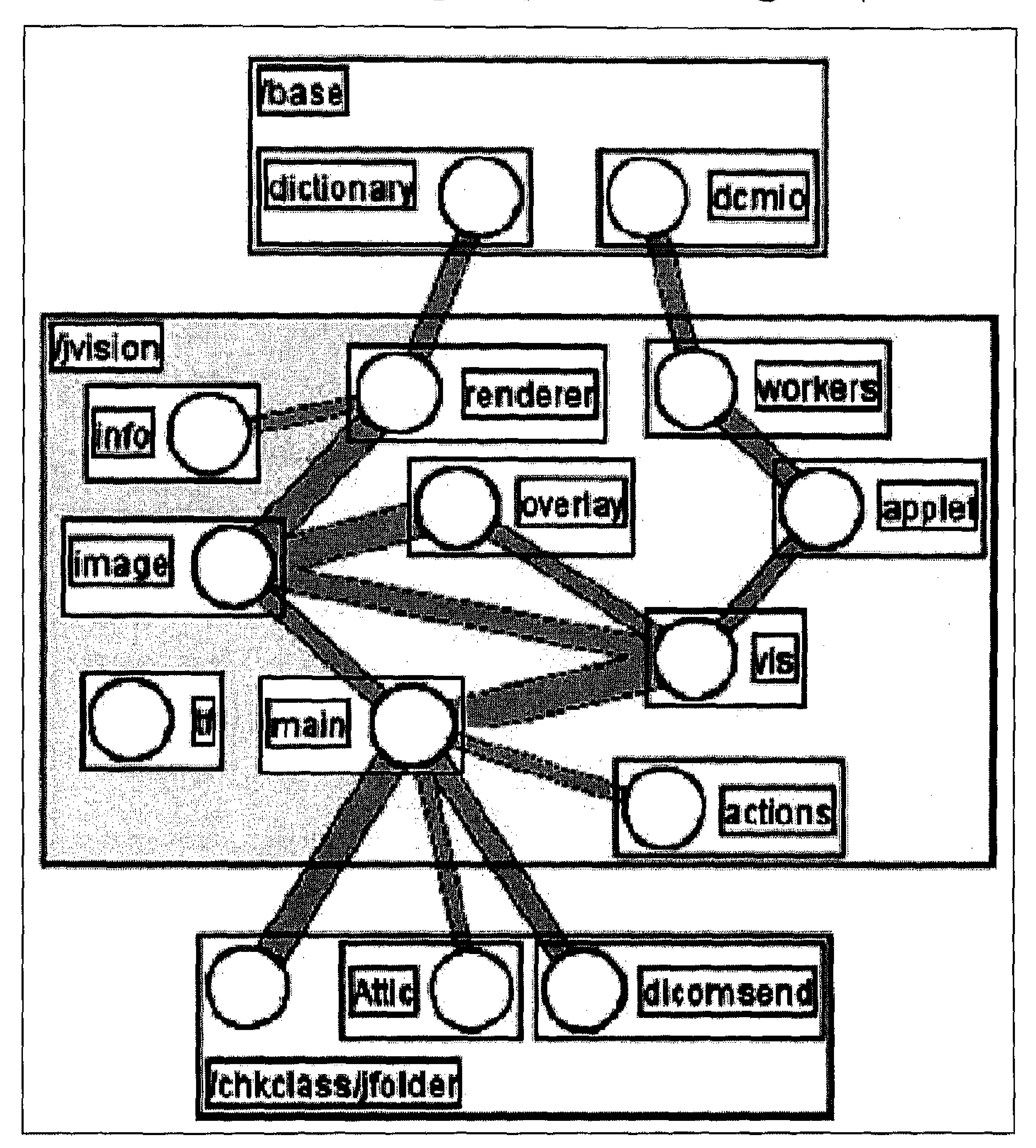
9 ـ 5 ـ 3 التصور الرسومي المتداخل

عند تحليل نظام برمجي كبير، يُنصح باتباع منهجية «التحليل من الأعلى إلى الأسفل» التي تبدأ من الصورة التفصيلية. تتبع منهجية EvoLens منهجية التحليل من الأعلى إلى الأسفل باستخدام الرسوم المتداخلة. يتم تصور الوحدات عند المستوى العلوي، أما الوحدات الفرعية فيتم تصورها في المستوى التالي يتبعها الملفات المصدرية التي يتم تصورها عند المستوى الأدنى. تُستخدم النقطة المركزية (البؤرة) لتركيز التحليل على علاقات تقارن التغيير لكيان معيّن. أما التوصيفات غير المتقارنة من حيث التغيير مع توصيف آخر فتهمل، وهذا ينتج مخططات أبسط وأسهل فهماً. النقطة المركزية معدّة من قبل المستخدم ليتم التركيز دائماً على التوصيف موضع الاهتمام. يوضح الشكل 9-11 مثالاً على عرض تطور يبيّن تقارنات التغيير بين الوحدة الرئيسة والوحدات الفرعية مع النقطة المركزية في الوحدة jvision.

يتم رسم النقاط كمستطيلات وتمثل الوحدات الرئيسة والوحدات الفرعية. يتم التعبير عن تداخل الوحدات الرئيسة والوحدات الفرعية بواسطة النقاط المتداخلة. على سبيل المثال، تبيّن النقطة التي تقع في المركز أن الوحدة jvision تتكون من 10 وحدات فرعية.

في منهجية EvoLens، يتم رسم العلاقات الداخلية وعلاقات التقارن بين الوحدات الداخلية. يشير عرض الخط إلى قوة التقارن من حيث عدد الاعتمادات المشتركة. عند مستوى الوحدة والوحدة الفرعية، يكون هذا العدد عبارة عن مجموع تقارنات التغيير الرئيسة بين الملفات المصدرية لأزواج الوحدات. المخطط الرسومي في الشكل 9-11 يشير إلى تقارنات التغيير بين الوحدات الفرعية التابعة للوحدة jvision. في هذه الوحدة، تتغير كل من image وmain

وrenderer وoverlay وvis كثيراً مع بعضها البعض، كما هو مبيّن بالخطوط السميكة المرسومة بين نقاط المخطط المترابطة. أما الوحدة الفرعية main فتسبب تقارن الوحدات الداخلية مع jfolder، وعليه فإن هناك مرشحاً أولياً لتغيير تصميم البرنامج من دون التأثير في النتائج refactoring.

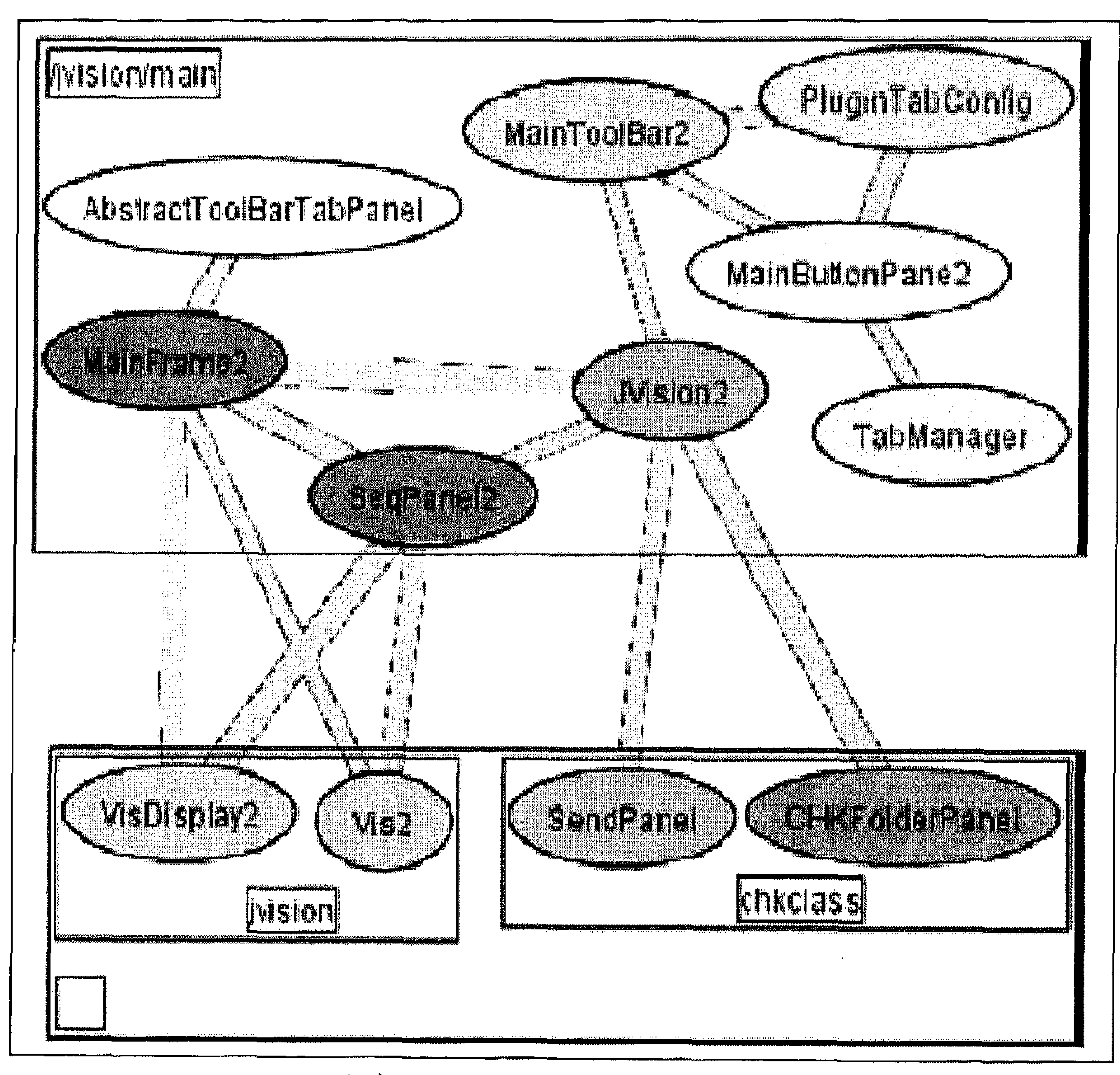


الشكل (9 ــ 11): مخطط متداخل لتصور الوحدة jvision[21]

يمكن إجراء التكبير والتصغير في المخطط الرسومي عن طريق توسيع أو طَيّ نقاط الوحدة أو الوحدة الفرعية. إضافة إلى آلية التكبير/ التصغير التقليدية،

توفّر منهجية EvoLens آلية فلترة ذات عتبات قابلة للتهيئة. وهذا يسمح للمُستخدم بالتركيز على التوصيفات ذات التقارن القوي، وتعمل على تصفية التوصيفات الأخرى ذات التقارن الضعيف. إضافة إلى نقطة التركيز (البؤرة)، يتم تقليل كمية المعلومات التي سيتم تصويرها في المخططات الرسومية.

على سبيل المثال، يشرح الشكل 9-12 مقطعاً مكبراً لمستوى الوحدة الفرعية jvision/main. انتقلت النقطة المركزية إلى الوحدة الفرعية main كما ظهرت محتوياتها ورسمت في مستطيل. في هذا المثال، النقاط المحتواة هي الملفات المصدرية التي يشير إليها الشكل البيضوي. يبيّن الشكل البيضوي ألواناً مختلفة تشير إلى تطور الملف. زاد الملفان MainFrame2 وSeqPanel2 أكثر من غيرهما، كما هو مبيّن باللون الأكثر كثافة للشكل البيضوي المناظر.



الشكل (9 ــ 12) : لقطة مكبرة لهيكلية الوحدة jvision[21]

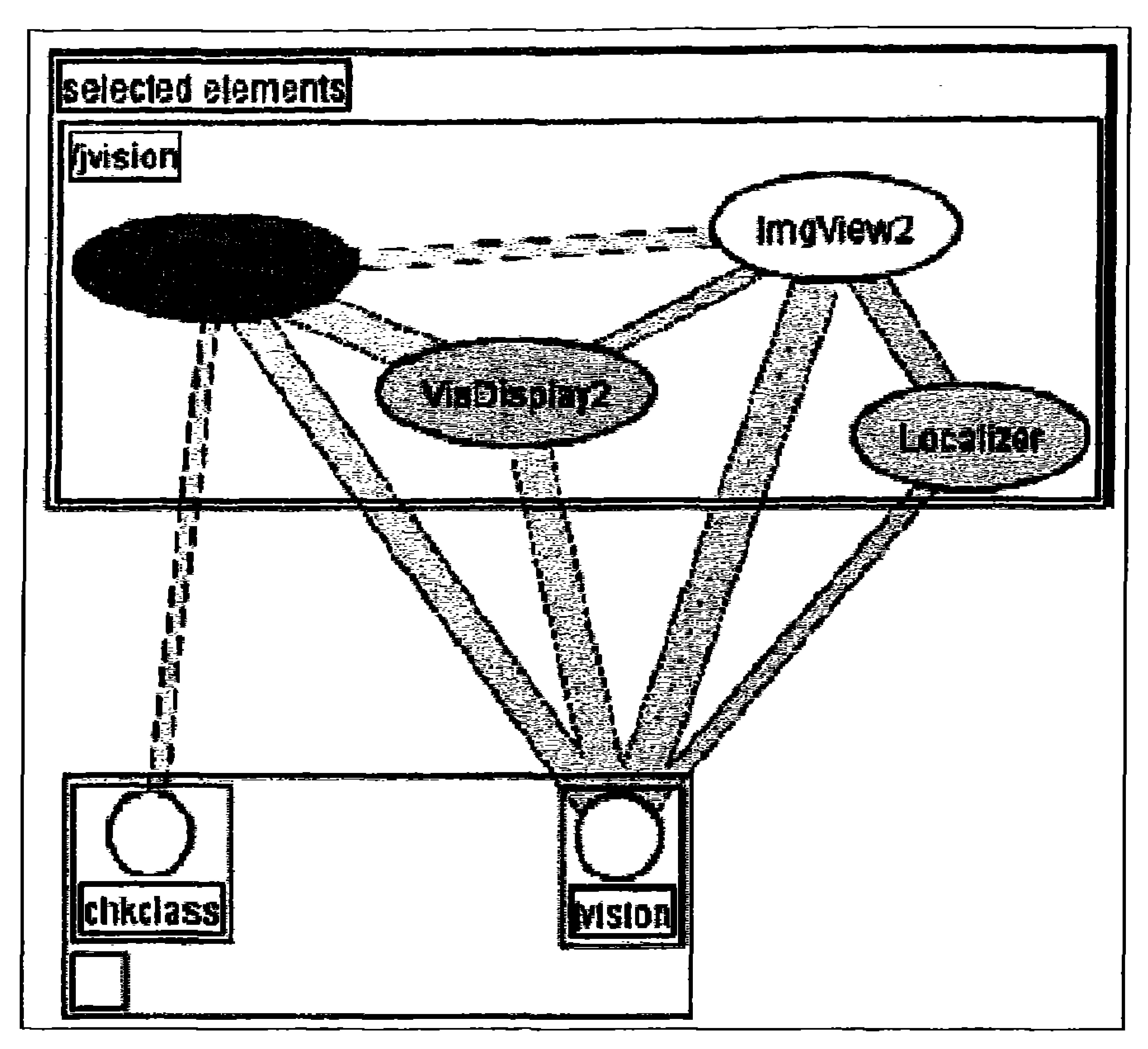
تُرسم تقارنات التغيير في الوحدات والوحدات الفرعية مع الوحدة الفرعية main كمستطيلات إضافية في المخطط البياني. في المثال، هذه الوحدات هي jvision مع الملفات المصدرية VisDisplay2 وVis2 والوحدة chkclass مع ملفين مصدريين هما SendPanel وCHKFolderPanel. يتم استثناء الملفات ذات تقارن التغيير الضعيف.

الملفات ذات تقارنات التغيير الضعيفة تُستثنى. بعد التشاور مع المطوّرين حول هذا العرض، وجدنا أن النوع الأساسي طبّق في JVision2. وهذه الحقيقة تبرر تقارن التغيير جزئياً لأن تهيئة الأجزاء الأخرى تتم عادة في أثناء البدء. لكن، أثمر الفحص التفصيلي للشيفرة البرمجية أن الوحدة JVision2 تفرض وصولية للعديد من أجزاء النظام من خلال متغيرات ثابتة. وهذا يجب أن يتم تحسينه.

9 ـ 5 ـ 4 تقارن التغيير الانتقائي

بما إن حدود الوحدة تكون مقيدة بشدة للتحقيق المتعمد في بعض الأحيان، تدمج EvoLens تصوّر مجموعات الملفات المختارة كلٌ على حدة. بإمكان المستخدم اختيار مجموعة الملفات في أثناء التحقق من البرمجية باستخدام الفأرة ويترك لـ EvoLens أمر بيان تقارنات التغيير لهذه المجموعة من الملفات. على سبيل المثال، في المخطط الرسومي الموضح في الشكل 9-13، اخترنا أربعة ملفات: MainFrame2 وVisDisplay2 وImgView2 وLocalizer. هذه الأنواع الأربعة هي تلك المسؤولة عن تقارنات التغيير القوية بين الوحدات الفرعية main وvis وimage والوحدة الرئيسة jvision. الملفات المختارة مجموعة في مستطيل، بينما تم طَيّ جميع الوحدات غير المختارة. هذا ويبيّن الشكل أيضاً تقارنات التغيير ضمن مجموعة الملفات وضمن الوحدات المطوية.

يبيّن المخطط الرسومي في الشكل 9-13 أن جميع الملفات الأربعة المختارة مقترنة من حيث التغيير بملفات الوحدة jvision. إضافة إلى ذلك، ثمة تقارنات تغيير ضعيفة بين MainFrame2 والأنواع التابعة للوحدة chkclass. وبمساعدة هذه الخاصية، يستطيع المُستخدم اختيار مجموعات مختلفة من الوحدات والوحدات الفرعية والملفات وتجميعها، ومن ثم تحليل تقارنات التغيير التي تربطها مع بقية النظام.

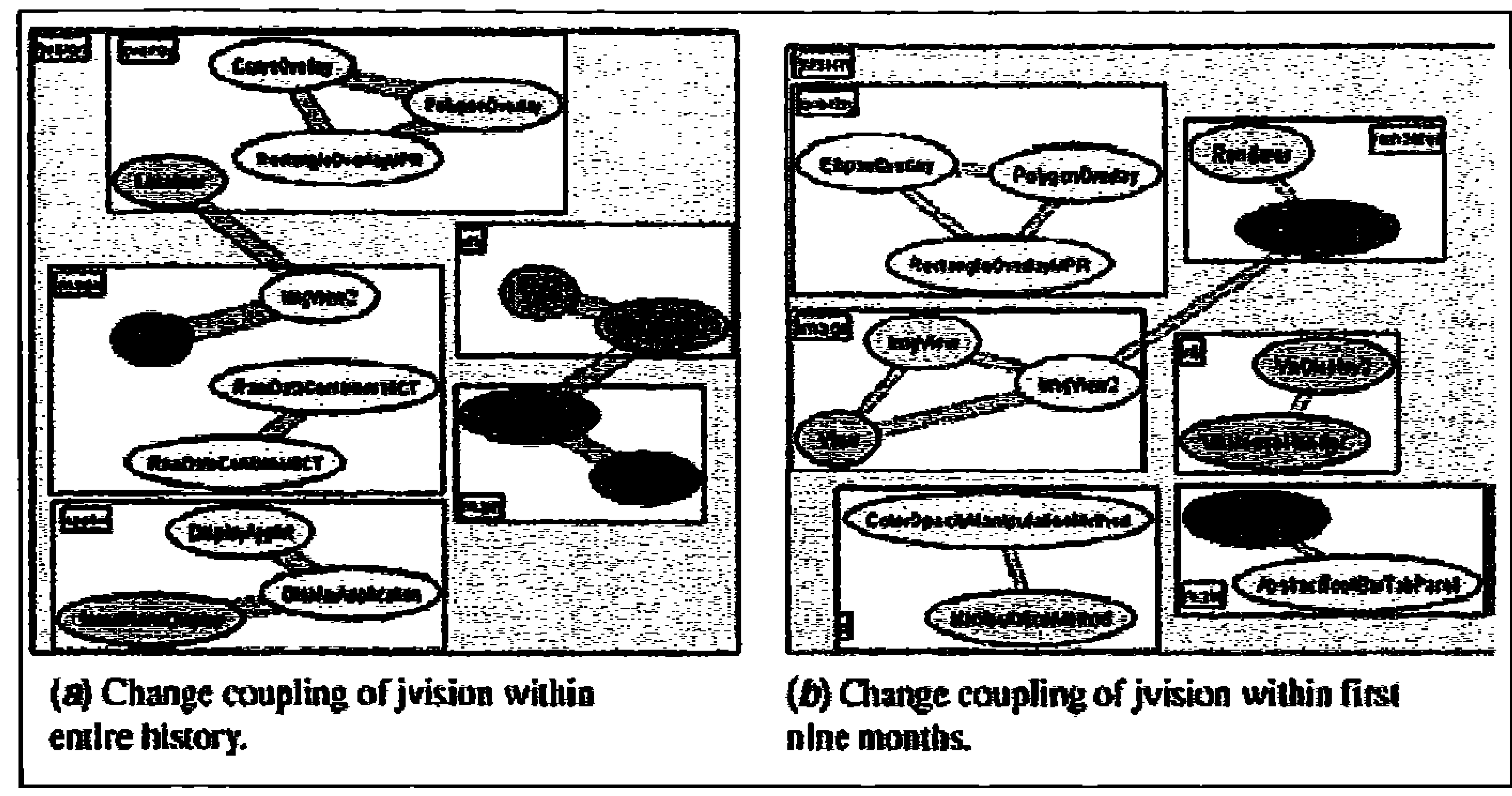


الشكل (9 ـ 13) : التقارنات الانتقائية بين الملفات MainFrame2 وVisDisplay2 وImgView2 وLocalizer .

تقاس تقارنات التغيير ضمن إطار زمني: عدد الاعتمادات المشتركة التي تمّت من خلال ملفين اثنين في أثناء فترة مراقبة معرفة. قد تكون فترة المراقبة محددة منذ بداية المشروع أو آخر ستة أشهر من المشروع على سبيل المثال. فترة المراقبة قابلة للتعديل من قبل المستخدم عن طريق تحديد زمن البداية والنهاية. عند تغيّر الفترة، تستجيب EvoLens عن طريق إعادة حساب علاقات تقارن التغيير، ومن ثم إعادة رسم المخطط.

يبيّن الشكل 9-14 الوحدات الفرعية لتقارن التغيير والملفات المصدرية للوحدة jvision خلال 18 شهراً من فترة التحقق أولاً، وأول 9 أشهر، ثانياً. يبيّن المخطط الموجود في الطرف الأيسر تقارنات التغيير التي لم تحدث خلال التسعة أشهر الأولى بين ملفين مصدريين هما MainFrame2 وVisDisplay2،

كما هو مبيّن في المخطط على الطرف الأيمن. نتيجة لذلك، تم تحديد التقارن بين الملفين لاحقاً خلال أنشطة التطوير.


(a) Change coupling of jvision within entire history.
(b) Change coupling of jvision within first nine months.

الشكل (9 ـ 14) : الشريحة الزمنية في EvoLens[21]

يوفّر لون الشكل البيضوي إشارات إضافية عن تطور الملفات المصدرية. على سبيل المثال، تقارن التغيير وألوان العرض وImgView2 أمر لافت.

ثمة علاقة تقارن تغيير قوية بين الوحدتين، لكن عرض الملف نما بعد إضافة ما يزيد على 300 سطر من الشيفرة البرمجية، في حين بقي الملف ImgView2 ثابتاً تقريباً عند 150 سطراً من الشيفرة البرمجية، لكن يتم تعديلها باستمرار. من الواضح أن النوع المنفذ في الملف ImgView يتطلب بنية أكثر ثباتاً ووضوحاً للحد من تأثير التغيير عند تعديل الملفات المرتبطة.

9 ـ 6 أعمال ذات علاقة

طوّر العديد من تقنيات وأدوات التصور في مجال الهندسة العكسية (أي إعادة التصميم) وفهم البرامج. من هذه الأدوات Bookshelf التي طورها فينيغان Finnigan وآخرون[4]، وDali التي طورها كل من كازمان Kazman و (13) كارير كارير Carriére، وBauhaus[2] التي طورها كوشكي Koschke وآخرون، وRigi

(2) <http://www.iste.uni-stuttgart.de/ps/bauhaus> .

التي طورها كل من مولر Müller وكلاشينسكي Klashinsky وCreole[3] التي طورها ستوري Storey وآخرون. تستخدم هذه الأدوات تقنيات تصور شبيهة بالرسم البياني لإنشاء عروض لإصدار شيفرة برمجية مصدرية معيّنة. تمثل النقاط (العقد) توصيفات الشيفرة البرمجية بينما تمثل الأطراف علاقات التبعية بين التوصيفات. هناك بعض الأدوات التجارية لإعادة التصميم وفهم البرامج. كما أن هناك بعض الأدوات التجارية لإعادة التصميم وفهم البرامج كـ Imagix4D وSotograph وCast. ترتكز تقنيات التصور الخاصة بناء على هذه التقنيات، لكنها تتضمن أيضاً بيانات عن التطور وإصدارات الشيفرة البرمجية المصدرية. هناك امتداد آخر لهذه التقنيات يتمثل في استخدام عروض القياسات المتعددة.

تستخدم تقنيات التصور خاصتنا العروض متعددة المقاييس التي عرفنا بها كل من Lanza ودوكاس Ducasse[16] وذلك لتصوّر الشيفرة البرمجية المصدرية مع مخططات رسومية غنية بقيم القياسات. قام الباحثان بدمج عدد من العروض المعرّفة مسبقاً في أداة CodeCrawler[15] مسهّلة بذلك تصور البرمجية بتفصيل. تتبع العروض مبادىء مطابقة القياس من حيث إن قيم القياسات الأكبر تؤدي إلى وجود أشكال أكبر حجماً في المخطط الرسومي.

استطاع Lanza والآخرون عرض وتقديم مصفوفة التطور[14] باستخدام مفاهيم العروض متعددة المقاييس وأخذ إصدارات الأنواع المختلفة في الاعتبار. بناءً على حجم القياسات التي تم تعقّبها من خلال عدد من الإصدارات، قام الباحثون بتعريف مفردات محددة لتصنيف الأنواع (مثلاً النابض، سوبرنوفا، القزم الأبيض. . . إلخ). بطريقة مماثلة، قام غيربا Gîrba وآخرون بوصف منهجية قائمة على تلخيص قيم قياس الشيفرة المصدرية لعدة إصدارات تحدد الأنواع المعرضة للتغير[9]. ثمة أداة تدعى العارض المصدري ثلاثي الأبعاد (sv3D) تستخدم استعارة metaphor ثلاثية الأبعاد لعرض النظم البرمجية وبيانات التحليل[17]. العرض التقديمي ثلاثي الأبعاد قائم على استعارة SeeSoft pixel[1] pixel[1] وتوسعها عن طريق تقديم التصورات في حيّز ثلاثي الأبعاد.

بالإضافة إلى تصور الشيفرة المصدرية، تم تطوير عدد من المنهجيات التي تركز على تصور الإصدارات وتاريخ التغيير للنظم البرمجية. على سبيل المثال،

---

(3) <http://www.thechiselgroup.org/creole>.

قام ريفا Riva وآخرون بتحليل ثبات الهيكلية[11, 8] باستخدام الألوان لوصف التغيرات التي أجريت خلال فترة مجموعة من الإصدارات. قام جينغوي وو Wu وآخرون بوصف طيف التطور[26] الذي يصور التسلسل التاريخي لإصدارات البرمجية. أما ريسلبيرغ Rysselberghe وديميير Demeyer فقد استخدما تصوراً بسيطاً قائماً على المعلومات المتوفّرة في نظم التحكم بالإصدارات لتوفير نظرة عامة عن تطور النظم[23]. أما فوانيا Voinea وآخرون، فقد قدموا منهجية CVSscan التي تتيح للمستخدم التحقق تفاعلياً من معلومات الإصدار والتغيير من مواقع التخزين في CVS من خلال عرض ذي توجه خطي. المنهجيات الأربع المقدمة في هذا الفصل تكمل التقنيات الموجودة وتوفّر وسائل إضافية لتحليل تطور النظم البرمجية كمخططات Kiviat أو الأشكال الكسيرية.

9 ـ 7 ملخص

إن عملية تحليل تطور النظم البرمجية هو نظرة إلى الخلف في تاريخ الإصدارات والتغيرات والعيوب والمشكلات. توفّر مواقع تخزين البرمجيات كـ CVS وBugzilla بيانات عن تاريخ الإصدار والتغيرات والمشكلات، إضافة إلى إصدارات الشيفرة البرمجية المختلفة التي يمكن استعادتها وقياسها من مواقع التخزين. يتم ضرب مقدار البيانات اللازمة لتحليل تطور البرمجية بعدد الإصدارات، وبالتالي يكون المقدار كبيراً. لمعالجة كمية البيانات هذه، تستخدم عروض عديدة وتقنيات تصور فاعلة.

في هذا الفصل، عرضنا أربع تقنيات للتصور، ركزت كل منها على جوانب مختلفة من تطور البرمجيات. يركّز عرض مقاييس التطور المتعددة على تصور الجوانب المتعلقة بتطبيق إصدار واحد من الشيفرة البرمجية المصدرية أو تطبيق عدة إصدارات.

على سبيل المثال، يمكن استخدام هذه التقنية لتصور تطور الوحدات من حيث الحجم ودرجة التعقيد مسلطة الضوء على الوحدات التي أصبحت غير مستقرة. يتيح عرض تطور الميزات للمطوّرين تعيين مواضع تنفيذ الميزات في الشيفرة البرمجية المصدرية وتقييم تأثير التعديلات الوظيفية في النظام البرمجي. يمكن أن تستخدم للتواصل مع المستخدمين والمطوّرين. أما لتحليل سلوك التغيير في الوحدات البرمجية، فيمكن استخدام عرض مساهمة المطوّرين. فهذه التقنية تصور الجهود المبذولة لإصلاح مشكلات النظام

وعيوبه وتطوير النظام البرمجي. هناك أربعة أنماط تطور تميّز سلوك تغيّر الوحدة التي تم تحديدها كالوحدات التي يعمل فيها عدد من المطوّرين بتوازن أو بطريقة غير متوازنة. أخيراً، تلقي تقنية عرض تقارن التغيير الضوء على الملفات المصدرية والوحدات التي تتغير مع بعضها البعض باستمرار. يمكن استخدام تصور هذه الاعتماديات المتقارنة المخفية في التحقق من جوانب القصور في التصميم (مثلاً، يجب أن تكون التغيرات محلية بالنسبة إلى الوحدة) ولتقييم تأثير التغيير.

بالنسبة إلى هذه التقنيات الأربع، عرضنا أمثلة من دراسات الحالة الخاصة بمشروع موزيللا ذي المصدر المفتوح بالإضافة إلى نظام برمجي صناعي. فهي تبيّن بوضوح فوائد المنهجيات وتقوية فكرة أن تقنيات التصور المختلفة لازمة لتحليل مظاهر التطور المختلفة للنظم البرمجية.

شكر وعرفان

يودّ المؤلفون تقديم الشكر لكلّ من Jacek Ratzinger وMarco D'Ambros وMichele Lanza لمساهمتهم في إنجاز هذا العمل.

ثم اعتمدت بعض أجزاء العمل المقدم من قبل «مؤسسة العلوم الوطنية السويسرية SNF» في إطار منحة المشروع «التحكم بتطور البرمجيات COSE» ومؤسسة Hasler ـ سويسرا في إطار منجة المشروع «فضاءات التصفح متعدد الأبعاد لتطور البرمجيات ـ EvoSpaces».

المراجع

1. T. Ball and S. G. Eick. «Software visualization in the large.» *IEEE Computer*: vol. 29, no. 4, 1996, pp. 33-43.

2. M. D'Ambros, M. Lanza, and H. Gall. «Fractal figures: Visualizing development effort for cvs entities.» paper presented at: *Proceedings of the 3rd International Workshop on Visualizing Software for Understanding and Analysis*, IEEE CS Press, Washington, DC, 2005, pp. 46-51.

3. N. E. Fenton and S. L. Pfleeger, editors. *Software Metrics: A Rigorous and Practical Approach*. 2nd ed. Boston, MA: International Thomson Computer Press, 1996.

4. P. Finnigan [et al.]. «The software bookshelf.» *IBM Systems Journal*: vol. 36, no. 4, 1997, pp. 564-593.

5. M. Fischer and H. Gall. «Visualizing feature evolution of large-scale software based on problem and modification report data.» *Journal of Software Maintenance and Evolution: Research and Practice*: vol. 16, no. 6, 2004, pp. 385-403.

6. M. Fischer, M. Pinzger, and H. Gall. «Analyzing and relating bug report data for feature tracking.» *Proceedings of the 10th Working Conference on Reverse Engineering*. Victoria, BC, Canada. IEEE Computer Society Press, Washington, DC, 2003, pp. 90-99.

7. M. Fischer, M. Pinzger, and H. Gall. «Populating a release history database from version control and bug tracking systems.» paper presented at: *Proceedings of the International Conference on Software Maintenance*, Amsterdam, Netherlands. Washington DC: IEEE Computer Society Press, 2003, pp. 23-32.

8. H. Gall, M. Jazayeri, and C. Riva. «Visualizing software release histories: The use of color and third dimension.» paper presented at: *Proceedings of the International Conference on Software Maintenance*, Oxford, UK; Washington, DC: IEEE Computer Society Press, 1999, pp. 99-108.

9. T. Gîrba, S. Ducasse, and M. Lanza. «Yesterday's weather: Guiding early reverse engineering efforts by summarizing the evolution of changes.» paper presented at: *Proceedings of the International Conference on Software Maintenance*, Chicago, IL; Washington, DC: IEEE Computer Society Press, 2004, pp. 40-49.

10. M. H. Halstead. *Elements of Software Science (Operating, and Programming Systems Series)*. New York: Elsevier Science Inc., 1977.

11. M. Jazayeri. «On architectural stability and evolution.» paper presented at: *Proceedings of the Reliable Software Technlogies-Ada-Europe*, Vienna, Austria. New York: Springer Verlag, 2002, pp. 13-23.

12. K. Kang [et al.]. Feature-oriented domain analysis (foda) feasibility study. Technical report, Software Engineering Institute, Carnegie Mellon University, 1990.

13. R. Kazman and S. J. Carriére. «Playing detective: Reconstructing software architecture from available evidence.» *Automated Software Engineering*: vol. 6, no. 2, 1999, pp.107-138.

14. M. Lanza. «The evolution matrix: Recovering software evolution using software visualization techniques.» paper presented at: *Proceedings of the International Workshop on Principles of Software Evolution*, Vienna, Austria. New York: ACM Press, 2001, pp. 37-42.

15. M. Lanza. «Codecrawler: Polymetric views in action.» paper presented at: *Proceedings of the 19th IEEE International Conference on Automated Software Engineering (ASE)* , Linz, Austria. Washington, DC: IEEE Computer Society Press, 2004, pp. 394-395.

16. M. Lanza and S. Ducasse. Polymetric views: «A Lightweight Visual Approach to Reverse Engineering.» *IEEE Transactions on Software Engineering*: vol. 29, no. 9, 2003, pp. 782-795.

17. J. I. Maletic, A. Marcus, and L. Feng. «Source viewer 3d (sv3d): A framework for software visualization.» paper presented at: *Proceedings of the 25th International Conference on Software Engineering*, Portland, Oregon. Washington, DC: IEEE Computer Society Press, 2003, pp. 812-813.

18. T. J. McCabe. «A Complexity Measure.» *IEEE Transactions on Software Engineering*: vol. 2, no. 4, 1976, pp. 308-320.

19. H. A. Müller and K. Klashinsky. «Rigi: A system for programming-in-the-large.» paper presented at: *Proceedings of the 10th International Conference on Software Engineering*, Singapore. Washington, DC: IEEE Computer Society Press, 1988, pp. 80-86.

20. M. Pinzger [et al.]. «Visualizing multiple evolution metrics.» paper presented at: *Proceedings of the ACM Symposium on Software Visualization*, St. Louis, Missouri. New York: ACM Press, 2005, pp. 67-75.

21. J. Ratzinger, M. Fischer, and H. Gall. Evolens: «Lens-view visualizations of evolution data.» paper presented at: *Proceedings of the International Workshop on Principles of Software Evolution*, Lisbon, Portugal. Washington, DC: IEEE Computer Society Press, 2005, pp.103-112.

22. D. F. Swayne, D. Cook, and A. Buja. Xgobi: «Interactive Dynamic Data Visualization in the X Window System.» *Journal of Computational and Graphical Statistics*: vol. 7, no. 1, 1998, pp. 113-130.

23. F. Van Rysselberghe and S. Demeyer. «Studying software evolution information by visualizing the change history.» paper presented at: *Proceedings of the 20th International Conference on Software Maintenance*, Chicago, Illinois. Washington, DC: IEEE Computer Society Press, 2004, pp. 328-337.

24. L. Voinea, A. Telea, and J. J. van Wijk. «Cvsscan: Visualization of code evolution.» paper presented at: *SoftVis '05: Proceedings of the 2005 ACM Symposium on Software Visualization*. New York: ACM Press, 2005, pp. 47-56.

25. N. Wilde and M. C. Scully. «Software Reconnaissance: Mapping Program Features to Code. *Journal of Software Maintenance*: vol. 7, no. 1, 1995, pp. 49-62.

26. J. Wu [et al.]. «Evolution spectrographs: Visualizing punctuated change in software evolution.» paper presented at: *Proceedings of the 7th International Workshop on Principles of Software Evolution*. Washington, DC: IEEE Computer Society Press, 2004, pp. 57-66.

الجـزء الرابـع

إدارة العمليـــات

10

الاختبار التجريبي في هندسة البرمجيات

جويسبي فيساجيو (Giuseppe Visaggio)

10 ـ 1 مقدمة

لمدة ما يقارب 30 عاماً في قطاع هندسة البرمجيات، كان الباحثون يطلبون من الممارسين استخدام نتائج أبحاثهم لتحويلها إلى ممارسات مبتكرة أو استخدام الممارسات التي طوروها في مختبرات الأبحاث للحصول على منتجات برمجية وعمليات مبتكرة، في حين كان الممارسون يطالبون بممارسات يمكن استخدامها للحصول على عمليات مبتكرة ومنتجات مدعمة بأدلة بهدف تقييم أهمية وجدوى العمليات والمنتجات والمخاطر التي يجب إدارتها في أثناء مأسسة العمل.

لسوء الحظ، يشير الباحثون عادة إلى المنافع التقنية لاستخدام عملية جديدة أو أداة جديدة، متجاهلين فائدتها وقابلية تحولها عملياً. بناءً على ذلك، يتطلب الممارسون فهرساً بنتائج البحث، وهذا ما لا يتحول على واقع عملي غالباً[33, 20] .

أيضاً، الممارسات المتوفرة لا تكون دائماً مدعومة بالأدلة اللازمة لضمان فعاليتها في العمليات الفعلية أو قابلية تحولها ضمن أو نحو عمليات الأعمال.

تساعد الأبحاث في مواجهة التوتر الناجم عن هذا الخلل والتغلب عليه عن طريق تطوير هندسة البرمجيات كعلم وإبعاده عن المظاهر التي جعلته يبدو أقرب إلى الفن منه إلى العلم[33] . في الواقع، يتطلب التطوير العلمي التحقق من النظريات والمبادئ والممارسات التي تكوّن المعرفة العلمية. إضافة إلى ذلك، قد تنتج الدراسات التجريبية دليلاً لشرح سبب أهمية تطبيق مبدأ جديد أو

تقنية جديدة (أي الابتكار). أخيراً، يمكن أن تسهم الدراسات التجريبية في بناء الاختصاصات التي يجب أن تتحول إلى ابتكار. ملخص ذلك أن الدراسات التجريبية قد تساعد في جعل هندسة البرمجيات جزءاً من المعرفة العلمية وتحويلها إلى ابتكارات ونشرها للآخرين.

يعطي هذا الفصل تركيباً للمعرفة التي جمعت في تصميم وتنفيذ الاستقصاءات التجريبية EI المدمجة مع الخبرة التي تكونت في مشاريع البحث المنفذة تحت إشراف المؤلف في مختبر أبحاث هندسة البرمجيات في جامعة Bari.

هذا الفصل مخصص لباحثي وطلاب هندسة البرمجيات الذين يرغبون في فهم دور البحث التجريبي في تنفيذ أنشطتهم وفي نقل نتائج أبحاثهم للمجتمع. على وجه الخصوص، تلك مساهمة مهمة في تدريس هندسة البرمجيات لفهم دور البحث التجريبي في تحديد زمن ومكان تقديم تقنية جديدة أو ابتكار تقنية مع تقنية أخرى. يجب أن يكون لدى قارئ هذا الفصل معرفة بعمليات وتقنيات وممارسات هندسة البرمجيات.

إضافة إلى ما تقدم، يتضمن هذا الفصل المخصص لهندسة البرمجيات الأقسام الآتية، تقدم الدراسات التجريبية مقدمة عامة عن:

(أ) الدراسات التجريبية ودورها في بناء النظريات والموافقة عليها.

(ب) وصف أنواع الاستقصاءات التجريبية المختلفة التي يمكن تنفيذها.

(ج) إرشادات عن تصميم وتنفيذ استقصاءات تجريبية فاعلة من دون عيوب.

بعد تحديد الجوانب الأساسية للاستقصاءات، يحلّل هذا الفصل استخدام هذه الاستقصاءات بهدف تحديد سمات هندسة البرمجيات وإمهارها ببصمة علمية. أما مخاطر وتهديدات الاستقصاءات التجريبية فهي حرجة وحاسمة من حيث التنفيذ الصحيح وتفسير البيانات. في هذا المعنى، يتضمن الفصل قسماً مخصصاً لصحة المخاطر والتهديدات، وهذا القسم يلخص معظم المخاطر ذات الصلة ويتضمن إرشادات لتنفيذ تجارب للتخفيف من أثرها. إضافة إلى ذلك، يقدم قسم الدراسات التجريبية في علم هندسة البرمجيات ما يأتي:

(أ) مقدمة عامة عن مسارات التطوير في هندسة البرمجيات والحاجة إلى تكرارها.

(ب) تحليل المشكلات التي تظهر في أثناء تكرار الدراسات وتوفير إرشادات للتغلب عليها.

(ج) تفاصيل عن أنواع معيّنة من التكرارات المستخدمة لاستخلاص المعرفة التي تعرف بعائلات التجارب.

يصف قسم الاستقصاءات التجريبية لقبول الابتكارات ما يأتي:

(أ) بعض التجارب المنفذة في مختبر أبحاث هندسة البرمجيات حول ملاءمة الدراسات التجريبية في المساهمة في القبول.

(ب) إضفاء الطابع المؤسسي على التكنولوجيا الجديدة.

أخيراً، بناء الاختصاصات من خلال الاستقصاء التجريبي يدعم فكرة أن الاستقصاء التجريبي في هندسة البرمجيات يجب أن يساهم في توسيع الاختصاصات، كما هو الأمر بالنسبة إلى العلوم الأخرى. في هذا المعنى، يلخص القسم كيف توسعت الاختصاصات خلال عشر أعوام من عمر الأبحاث التجريبية في هذا النطاق. أما خاتمة القسم فتختم الفصل العاشر مع الاعتبارات التي تشير إلى القضايا التي لا تزال مفتوحة التي تتبع ما تم عرضه في الأقسام السابقة من الفصل.

10 ـ 2 الدراسات التجريبية

10 ـ 2 ـ 1 مقدمة عامة

يبدأ التطوير العلمي بـ (أ) مشاهدة حدث معيّن ويهدف إلى صياغة قوانين ملزمة عالمياً، و(ب) وضع نظريات تجريدية. الاستقصاء التجريبي بمعناه الواسع هو رديف للدراسات الميدانية وهو عملية تهدف إلى اكتشاف شيء مجهول أو التحقق من صحة فرضية يمكن تحويلها عموماً إلى قوانين صحيحة شرعية. يشارك الباحث في استخراج وجمع البيانات التي تحدد مقدار المشاهدات اللازمة. يمكن أن تشمل المشاهدة تأكيداً ذاتياً؛ عموماً، تتم المشاهدات بالاعتماد على حواس الباحث كلها. ثمة أدوات كأجهزة الاستشعار تتيح لنا إجراء المشاهدات من دون استخدام الحواس. غالباً ما تكون هذه الأدوات إلكترونية وتمنح الباحث قدرات للمشاهدة.

تُجرى بعض المشاهدات صدفةً؛ بينما يكون البعض الآخر دورياً. يكون الباحث مهتماً بالنوع الأخير من المشاهدات. يهدف بحثه إلى اكتشاف الأسباب

التي تحدد الآثار المُلاحَظة على الأحداث. إضافة إلى ذلك، إذا كانت الأحداث إيجابية، يجب أن تكون المسببات التي نتجت منها تلك الأحداث معروفة وذلك بهدف تكرارها. أما إذا كانت سلبية، فيجب أن تكون المسببات معروفة أيضاً، لكن بهدف تجنّب مثل هذه الأحداث.

في البداية، المسبّبات التي تحدّد الأحداث الملاحظة تحدّد الفرضية. والفرضية يمكن أن تكون مقبولة مبدئياً عندما تكون مدعومة بسبب منطقي يمكن اعتباره مقبولاً بالنظر إلى المعرفة الحالية في مجال البحث نيابة عن المجتمع العلمي، يمكن أن يرفض الباحثون الفرضية. الفرضية تبقى «حدساً» إلى أن يقبلها العديد من الباحثين. حالما تُشكّل الفرضية/ الحدس، يقوم الباحثون المتحمسون لها بإجراء الاستقصاءات التجريبية لدعمها وإثباتها. على العكس من ذلك، يجري الباحثون الذين لا يشاركون في تلك الفرضية استقصاءات تجريبية لرفضها.

في بعض الحالات، علاقة السبب ـ التأثير المؤكدة بفرضية أو حدس، على الرغم من أنها غير مدعومة بأي دليل، تحدث في حالات حقيقية كثيرة. هذه الخبرة يتناقلها الممارسون عندما تلبي الممارسات أهدافهم. في هذه الحالة، يسمى الحدث أو الفرضية «حدساً مهنياً»(*).

تصبح الفرضية قانوناً عندما تُدعم بدليل تجريبي راسخ. أما إذا لم يتم إثبات الفرضية بدليل كافٍ حتى تعتبر صالحة، يقال إنها تؤدي إلى مبدأ وليس قانون. يلخص القانون/ المبدأ المعرفة التي جمعها الباحثون في أثناء الاستقصاء التجريبي وهو وسيلة لتحويل هذه المعرفة إلى عمليات أعمال. لذا، يتم تحليل البيانات التي جمعت في أثناء المشاهدات بهدف التحقق منها ودعمها أو دحض النظرية التي تمنع الفرضية[24].

يشرح القانون كيفية حدوث الحدث ضمن ظروف معيّنة، لكنه لا يشرح سبب الحدث. أما سبب حدوث بعض الأحداث ضمن ظروف محددة فتوضحه النظرية. أحياناً، تظهر النظرية قبل القانون؛ في هذه الحالة، يمكن للنظرية التوقع بالقانون والمشاهدات. لكن، يجب أن يثبت الباحث النظرية من خلال الاستقصاء التجريبي والمشاهدات.

(*) الحدس المهني (Heuristic): مصطلح يشير إلى الطرق المستخدمة في حل المشاكل الإنسانية والآلية باللجوء إلى التجارب أو الخبرات التقنية. كما يعرّف معجم اللغة العربية بالقاهرة الحدس المهني بأنه المقدرة النامية للفرد لحل المشاكل عن طريق الخبرة الطويلة (المترجم).

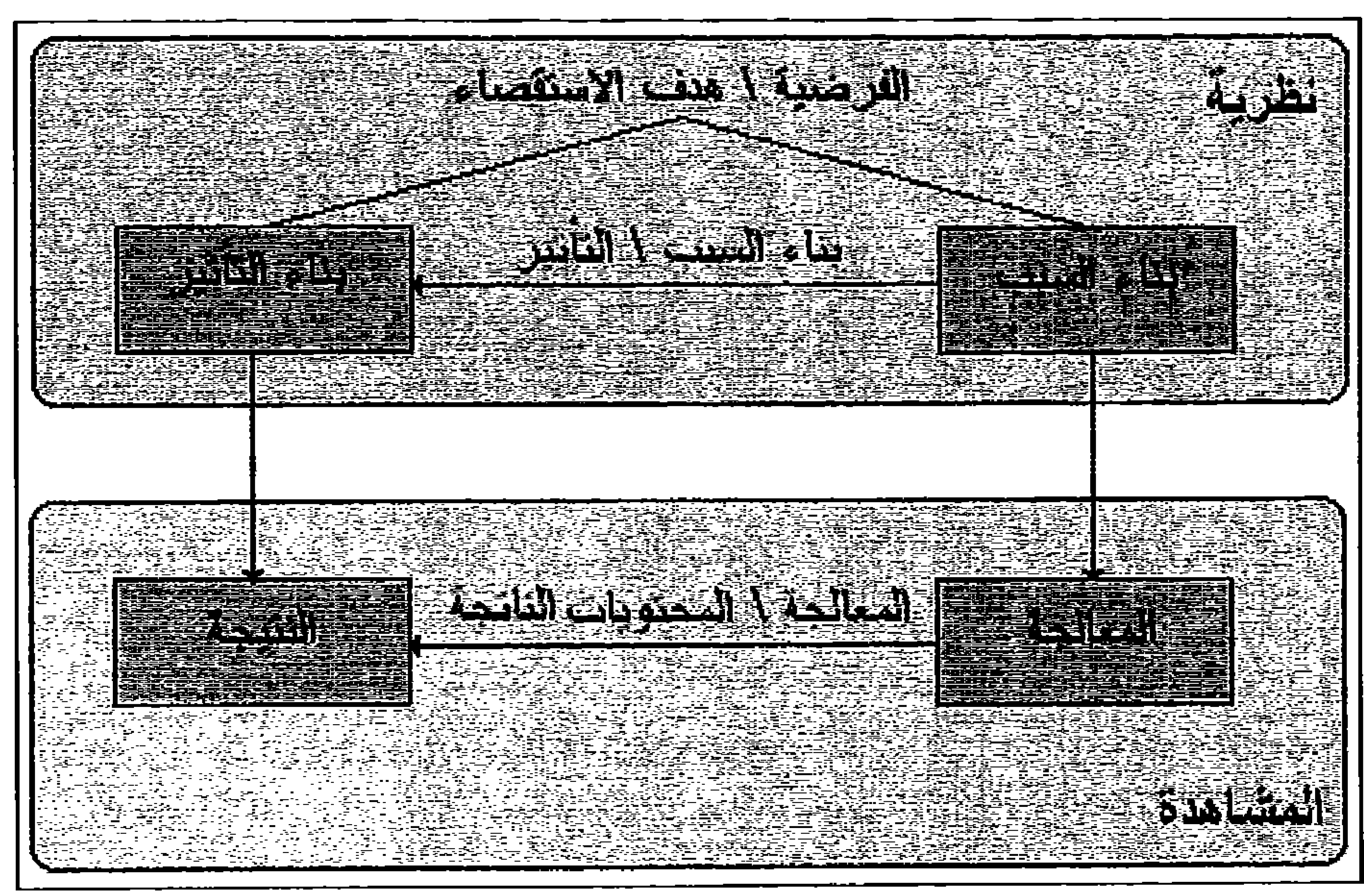
يمكن طرح مثال بالإشارة إلى قانون Parnas حول إخفاء المعلومات[32]. لاحظ بارناس أن ما قام به المطوّرون في شركة فيليبس (Philips) طَبّق حدساً مهنياً تقليدياً على تصميم البرمجية: فقد كانت كل وحدة مصممة ذات أبعاد صغيرة، ويمكن فهمها وبرمجتها بشكل مستقل عن المصمّم؛ وكانت واجهات الاستخدام بسيطة بما فيه الكفاية لشمولها في نطاق العمل وقد صممت بصورة مستقلة عن المصمم أيضاً. افترض بارناس أنه لتحسين إمكانية صيانة النظم البرمجية، يتوجب تنفيذ كل وحدة بشكل مستقل عن الوحدات الأخرى، وكانت الواجهات تعرض أقل مقدار من المعلومات الممكنة في ما يتعلق بتنفيذها. لإنجاز نطاق العمل، وكانت التقنية المقترحة ما يأتي: بإعطاء قائمة بقرارات المشروع، أخفي كل قرار في وحدة. كانت القرارات ذات احتمالية التغيير الأكبر تُنفّذ في وحدات ذات علاقات قليلة مع الوحدات الأخرى في النظام. هذه التقنية لم تعد تعتبر أن أداء النظام ذو أهمية أساسية. يجب التحقق من الأداء بعد إنتاج البرمجية وذلك بإجراء اختبارات معيّنة، فإذا انتهك الأداء، يجب أن يتخذ المصمم قرارات جديدة تخفى في وحدات جديدة لتغيير إصدار النظام البرمجي السابق. هذه المعرفة نظرية في ما يعرف بقانون Parnas: وملخصها أن كل شيء مخفي يمكن تغييره من دون تحمّل مخاطرة.

لم ينفّذ بارناس أي دراسات تجريبية ليمنح هذا القانون الصفة الشرعية. استخدم العديد من الباحثين هذا القانون لكن لم يقم أي منهم بتكراره أو نشر نتائجه المهمة لإثبات صحته وإمكان تعميمه عالمياً. اذاً، من الأصح أن يسمى «مبدأ» بدلاً من «قانون».

يمكن تمرير النظرية الكامنة في هذا القانون كما يأتي. بإعطاء المحاذير الآتية، إخفِ قرارات المشروع البرمجي في وحدة ووحدات تصميم بحيث (أ) تكون محتوياتها مستقلة عن بعضها البعض، و(ب) واجهاتها لا تعتمد على تغيّر محتويات الوحدة. التأثير الناتج هو كما يلي: يرتبط كل تعديل في المتطلبات بتغيير في أحد القرارات أو أكثر، وبالتالي تغيير في إحدى الوحدات أو أكثر؛ المتابعة بين القرارات والوحدات تتيح لنا تحديد التأثير الأولي في التعديلات؛ يضمن ثبات الواجهة أن التعديلات التي تجرى على كلّ وحدة متغيرة لا تؤثر في غيرها من الوحدات المرتبطة بها؛ إن استقلالية المحتويات بين الوحدات يتطلب أن يكون المطوّر هو الوحيد الذي يعرف محتويات الوحدة المتغيرة. شكراً لهذه الكمية المحدودة من المعرفة المتطلبة، الصيانة هي الأكثر سرعة

(يمكن تنفيذ التعديلات بالتوازي من قبل عدد من المطوّرين موافق لعدد الوحدات التي يجب تعديلها من دون أن تتداخل أعمالهم مع بعضهم البعض)، التكلفة ذات تأثير (التأثير الأولي للتعديلات يتناسب مع نطاق العمل أو مع التغيير المطلوب، بينما يكون التأثير الثانوي محدوداً)، والموثوقية (تحدّد الوحدات التي يتم تعديلها من خلال طلبات تغيير محددة).

من وجهة نظر الاستقصاء التجريبي (EI): إن المظهر المركزي الذي يمكن اعتباره اقتراحاً يُعبّر عن علاقة «سبب ـ تأثير» بين بنيتين نظريتين، يؤسس إلى بنية سببية تعبّر عن أسباب ناجمة، وبنية تأثير تصف التأثيرات المشتقة. وتربط النظرية بين الفرضيات بواسطة بُنى محكمة[44]، كما هو مبيّن في الشكل 10-1.



الشكل (10 ـ 1): مفاهيم أساسية للاستقصاء التجريبي وعلاقاتها مع بعضها البعض

يهدف الاستقصاء التجريبي إلى ملاحظة تأثير التغيير على أحد العوامل أو أكثر. المشاهدات التي يتم تنفيذها مع الاستقصاء هي نتيجة المعالجات؛ كل عملية معالجة تُميّز بقيمة محددة لكل عامل من العوامل. يتم التعبير عن هذه العوامل من خلال متغيرات مستقلة. القيم التي يمكن أن تفترضها المتغيرات المستقلة تسمى «مستويات». ينتج من تنفيذ أنشطة الاستقصاء تأثير أو أكثر يعبّر عنها بالمتغيرات التابعة التي تكون نتائج الاستقصاء. يجب أن يتم تنفيذ

الاستقصاء التجريبي في سياق موصوف بمجموعة من الخصائص التي يجب أن لا تتغير في أثناء الاستقصاء؛ وتسمى هذه الخصائص بالعوامل. عندما يكون لأحد العوامل أو أكثر تأثير في استجابتها، وهذا أمر لا يهمنا، فإن قيمها تحجب؛ تسمى مثل هذه المتغيرات بالعوامل المانعة. نأخذ بالحسبان الأشخاص الذين يعملون في مجموعة أو كتلة ونعتبرهم بقيمة العوامل المانعة نفسها، بحيث نتمكن من دراسة تأثير معالجة كتلة فقط. إذا كان لدينا العديد من الكتل ذات قيم مختلفة عن العامل المانع نفسه، فإنه لا يمكن دراسة التأثير بين الكتل. يتم تنفيذ المهام في أثناء الاستقصاء من قبل المطوّرين كعيّنات تجريبية. إن تحقيق هدف الاستقصاء يشكّل وحدة تجريبية.

على سبيل المثال، افترض أننا نتحقق ممّا إذا كان أسلوب التصميم الجديد ينتج برمجية بجودة أعلى من أسلوب التصميم المستخدم مسبقاً. يتم تنفيذ تجربة بأسلوب التصميم الذي له عامل بمستويين: أسلوب التصميم الجديد والأسلوب القديم. كل مستوى يُعتبر معالجة. أما المسؤولون التجريبيون فهم المصممون ذوو الخبرة المحددة مسبقاً. أما العوامل المتغيرة فقد تكون المتغيرات التي تعبِّر عن الخبرة كالمصمم الذي يعمل مع المسؤولين التجريبيين. الوحدة التجريبية هي طريقة تطوير البرمجية في سياق الشركة س (XYZ). المتغير التابع قد يكون عدد الأخطاء (العيوب) الموجودة في التطوير. إذا استخدم بعض المصممين إحدى طرق التصميم (أي التصميم كائني التوجه)، فإن بإمكانهم جمعها في مجموعتين، واحدة بوجود خبرة كالخبرة في التعامل مع العامل المانع أو خبرة في هذا الأسلوب المستخدم، وأخرى من دون خبرة بهدف التقليل من تأثير مثل هذه الخبرة.

10 ـ 2 ـ 2 استراتيجيات الاستقصاءات التجريبية

في البحث التجريبي، يكون الباحث مسيطراً على بعض ظروف التنفيذ وعلى المتغيرات المستقلة التي يجب دراستها. حسب الحالة التي ينفّذ فيها الاستقصاء، يمكن أن يتم استدعاؤها في بيئة حيوية in vivo(*) إذا كان الاستقصاء ينفّذ على عمليات الإنتاج ضمن مشروع قيد العمل أو في

(*) في بيئة حيوية (باللاتينية in vivo)، يطلق هذا المصطلح على العمليات أو التعديلات التي تحدث في نفس بيئة التشغيل (المترجم).

بيئة المختبر in vitro(*) إذا كان ينفّذ في بيئة مختبر معزولة.

بناءً على مستوى التحكم بالمتغيرات، يمكن تصنيف الاستقصاءات كما يأتي: المسح (الدراسة)، تحليل ما بعد الحدث (يعرف أيضاً بتحليل بأثر رجعي)، استقصاء المشروع، أو التجربة المسيطر عليها (أو تعرف بالتجربة).

المسح هو دراسة تجريبية يتم فيها جمع البيانات من الأطراف موضع البحث من خلال الاستبيانات أو المقابلات. وقد تكون بأثر رجعي حيث تجرى على الخبرات المجمعة من الأشخاص الذين استخدموا الأدوات أو الأساليب أو العمليات المستهدفة في الاستقصاء، أو قد تكون دراسة تمهيدية، أي إنها تجمع الآراء عن الابتكارات التي يجري الاستقصاء عنها مسبقاً.

مثال على ذلك، من المنطقي بيان مسح حول رضا الطلاب عن مقرر هندسة البرمجيات أجرته إدارة المعلوماتية في جامعة Bari. أجري هذا المسح عن طريق توزيع استبيان على طلاب المادة بعد أن تقدموا للامتحان. أجاب الطلاب حسب خبراتهم ومعرفتهم التي حصّلوها في أثناء الفصل الدراسي. أظهر تحليل بيانات الإجابات ما إذا كانت التغيرات التي أجريت في أثناء الفصل الدراسي قد اعترف بها الطلاب إيجابياً أم إذا كان لهم عليها بعض الملاحظات والتعليقات. في ما يلي، أخذت البيانات الرقمية في التقارير فقط؛ كما تضمن الاستبيان ملاحظات نصّية حفّزت الطلاب على الإجابة. لم تؤخذ الملاحظات في التقرير لأسباب تتعلق في المساحة. لفهم الاستبيان أفضل ما يكون، من الأهمية بمكان الإشارة إلى أن الطلاب قاموا بتنفيذ مشروع قاموا باختياره من بين مجموعة من المقترحات. بعض أعمال المشروع اقترحها الشركاء في القطاع الصناعي وكانت تتضمن مقترحات لا تشير مباشرة إلى محتويات المادة الدراسية، بل إلى الكفاءة المكتسبة في أثناء الفصل الدراسي من المادة. تعرف تلك الأعمال بأعمال المشروع الصناعية.

أما بعض مقترحات المشروع الأخرى فقد قدمها مدرسو المادة، وكان محتوى المقترحات يشير إلى محتويات المادة مباشرة. وتعرف تلك الأعمال بأعمال المشروع التعليمية. يبيّن الجدول 10-1 الاستبيان، في ما يوضح الشكل 10-2 النتائج المحصلة بعد توزيع الاستبيان.

(*) في بيئة المختبر (باللاتينية in vitro، ومعناه حرفياً في الزجاج)، يطلق هذ المصطلح على إجراء العملية في المختبر خارج البيئة الأصلية حيث هناك سيطرة أكبر نوعاً ما (المترجم).

تضمنت آراء الطلاب مقترحات بأن يقوم المدرسون بتحسين محتوى ومضمون المادة وكيفية توفيرها. من الجدول 10-2، يمكننا أن نرى أن مستوى الرضا بقي ثابتاً بعد التحسينات التي اقترحها الطلاب.

يتكون استقصاء تحليل ما بعد الحدث (بتحليل بأثر رجعي) من تحليل البيانات المجمعة في البيئة الحيوية in vivo لكن لأهداف تختلف عن تلك الاستقصاءات التي تتم في المشاريع المنفّذة أصلاً. فهذا النوع من الاستقصاءات يستخدم لتأكيد أو نفي إحدى الفرضيات (أو أكثر) عن طريق التحقق من أن البيانات التي جمعت من المشاريع المنفّذة قادرة على محاكاة متغيرات الاستقصاء. لهذه الطريقة مخاطر منخفضة وذات تكلفة منخفضة وتضمن صلاحية تجريبية منخفضة أيضاً. عادة ما تكون الأدوات الإحصائية عبارة عن تحليل سلاسل البيانات التاريخية والتنقيب عن البيانات.

مثال على ذلك الاستقصاء بأثر رجعي عن إطار عمل متعدد العروض لتصميم خطة قياس هدفية التوجه. هنا، نوفر المحتويات الضرورية للاستقصاء؛ بإمكان القرّاء المهتمين الرجوع إلى المرجع[5]. على الرغم من وجود دليل على التطبيق الناجح لبرامج القياس هدفية التوجه في القطاعات الصناعية، إلا أنه لا يزال هناك العديد من القضايا المفتوحة: أبعاد خطة القياس وتعقّد التفسيرات والاعتماديات بين الأهداف والمواعيد الزمنية لأنشطة القياس. تتضمن نظريتنا بناء هيكلية برنامج القياس بحيث تكون الأهداف متماسكة داخلياً من حيث عدد ونطاق المقاييس بينما تكون علاقاتها مع بعض البعض ضعيفة وذلك لإدارة تعقد التفسيرات. حتى تكون هذه النظرية فعالة، نقترح منهجية إطار العمل متعدد العروض MF التي توفر دعماً لتصميم خطة قياس صناعية ذات هيكلية كبيرة. أساسياً، صنّفت أهداف تحاليل منهجية إطار العمل متعدد العروض كما يأتي: أهداف العملية (Prosses Goals) ـ تنحو إلى تقييم عامل الجودة للفائدة قياساً بالطريقة أو الفعالية؛ أهداف إدارة المشروع (Project Management goals) PM ـ تنحو إلى تقييم النشاطات اللازمة للتخطيط لتنفيذ المشروع والتحكم به؛ كفاءة أهداف الاستثمار (Fitness of Investment Goals) FI ـ تقييم المظاهر كنسبة التكلفة ـ المنفعة والمخاطر المحتملة. يعرّف كل مظهر من هذه المظاهر كعرض.

يتم الاستقصاء عن هيكلية خطة القياس عن طريق تتبع الأهداف باستخدام جدول الإسناد الترافقي لعرض الأهداف. يمكن تعريف ذلك بوضع الأهداف في

أسطر الجدول ووضع العروض كأعمدة (انظر الجدول 10-3). إذاً، فإن كل مدخل (X) في الخلية (i,j) في الجدول يعني أن الهدف ذا الترتيب ith له مقاييس ترتبط بالعرض ذي الترتيب jth.

الجدول (10 ـ 1)

استبيان

السؤال	الخيارات
1. ما هو رأيك في محتوى المادة؟ ما هي المواضيع التي يجب دعمها بمزيد من التفاصيل؟ ————————————————	☐ متوازن جيداً ☐ غير متوازن
2. ما هي دراسات الحالة التي نفذتها؟	☐ دراسة حالة عن القطاع الصناعي ☐ دراسة حالة تعليمية
3. برأيك، ما هو أكثر الخيارات فعالية للمادة؟ (يمكن انتقاء أكثر من خيار). حدد إذا اخترت "غير ذلك": ————————————	☐ دراسة حالة تعليمية ☐ دراسة حالة عن القطاع الصناعي ☐ تمارين مخبرية ☐ تمارين صفية ☐ أسئلة مكتوبة ☐ غير ذلك
4. ما هو رأيك في دراسة الحالة عن القطاع الصناعي؟ (يمكن انتقاء أكثر من خيار). عزز إجابتك: ————————————	☐ تحسن من مستوى الاحترافية ☐ تتطلب مجهوداً كبيراً ☐ تجربة سلبية
5. بشكل عام، هل تعتبر أن المادة: (يمكن انتقاء أكثر من خيار). عزز إجابتك: ————————————	☐ شاقة ☐ مفيدة
6. صف ثلاثة منافع للمادة: 1. ———————————— 2. ———————————— 3. ————————————	
7. صف ثلاثة جوانب ضعف للمادة: 1. ———————————— 2. ———————————— 3. ————————————	

الجدول (10 ـ 2)
نتائج المسح

	عدد الاستبيانات الموزعة		143		76		63		50	
			2002 ـ 2003		2003 ـ 2004		2004 ـ 2005		2005 ـ 2006	
	السؤال	الإجابة	القيمة	التباين	القيمة	التباين	القيمة	التباين	القيمة	التباين
1	محتوى المادة	متوازن جيداً	٪88.11	-	٪93.42	0.05	٪87.30	-0.07	٪84.00	-0.04
		غير متوازن	٪11.89	-	٪6.58	-0.8	٪12.70	0.48	٪16.00	0.2
2	دراسة الحالة المنفذة كانت . .	دراسة حالة عن القطاع الصناعي	٪93.71	-	٪88.16	-0.06	٪92.06	0.04	٪98.00	0.06
		دراسة حالة تعليمية	٪6.29	-	٪11.84	0.46	٪7.94	-0.5	٪2.00	-2.97
3	أكثر الخيارات فعالية للمادة هو: (يمكن انتقاء أكثر من إجابة)	دراسة حالة تعليمية	٪21.68	-	٪19.74	-0.09	٪15.87	-0.24	٪18.00	٪0.12
		دراسة حالة عن القطاع الصناعي	٪90.91	-	٪86.84	0.05 ـ	٪93.65	0.72	٪86.00	0.08 ـ
		تمارين مخبرية	٪10.49	-	٪17.11	0.38	٪6.35	-1.69	٪18.00	0.65

يتبـــع

تابــع

		تمارين صفية	%20.28	-	%23.68	0.14	%15.87	-0.5	%10.00	-0.6
		أسئلة مكتوبة	%9.79	-	%10.53	0.07	%4.76	-1.21	%12.00	0.6
		أخرى	%0.00	N/A	%0.00	N/A	%0.00	N/A	%0.00	N/A
4	رأيك في دراسة الحالة عن القطاع الصناعي؟ (يمكن انتقاء أكثر من خيار)	تحسن من مستوى الاحترافية	%97.90	-	%98.68	0.008	%92.06	-0.72	%98.00	0.06
		تتطلب مجهوداً كبيراً	%56.64	-	%53.95	-0.05	%55.56	0.03	%72.00	0.23
		تجربة سلبية	%1.40	-	%0	N/A	%6.35	N/A	%6.00	-0.06
5	بشكل عام، كانت المادة : (يمكن انتقاء أكثر من خيار).	شاقة	%72.73	-	%63.16	-0.15	%68.25	0.07	%76.00	0.10
		مفيدة	%98.60	-	%100.00	0.014	%95.24	-0.05	%88.00	-0.08

تمّ التحقق من صحة الجدول من خلال مؤشرين: عدد التبعيات #Dependencies وهذا العدد يزيد كلما زاد عدد العروض التي تؤخذ بعين الاعتبار. ويتم حساب هذا المؤشر بالمعادلة الآتية:

$$\#\text{Dependencies} = \sum_{i=1}^{k} (N_i - 1)$$

حيث k، إجمالي عدد الأسطر في جدول الإسناد الترافقي، وN_i عدد مرات X في جدول الإسناد الترافقي، مع الإشارة إلى السطر ذو الترتيب i^{th}.

الجدول (10 ـ 3)

الإسناد الترافقي لعرض الأهداف

	العملية				المنتج				
الأهداف	ع1	ع2	ع3	ت0	ت1	ت2	ت3	إدارة المشروع	كفاءة الاستثمار
هـ 1	X				X				
هـ 2		X							
هـ 3					X				
هـ 4								X	X
هـ 5			X				X		

المؤشر الثاني هو كثافة التبعيات، ويقيّم مقدار قوة التبعيات؛ يتم حسابها حسب المعادلة الآتية:

$$\text{كثافة التبعيات} = \frac{\text{عدد التبعيات}}{(\#\text{الأسطر} \times \#\text{الأعمدة}) - \#\text{الأهداف}}$$

خطة القياس المصممة جيداً يجب أن تحدّ من كلا المؤشرين.

للتحقق من صحة إطار العمل متعدد العروض، يجب استخدام خطة قياس متوافرة، تم تحديد الخطة ضمن مشروع منفّذ، ومن ثم طبقت المنهجية المقترحة عليها. كان التحقق قائماً على تحليل كيف كانت هيكلية خطة القياس ستكون لو استخدمت منهجية إطار العمل متعدد العروض لتصميمها. أولاً، تم تنفيذ التحقق المتقاطع. ثم عقدت مقارنة بين جداول الإسناد الترافقي لعرض الأهداف لخطة

القياس القديمة غير المهيكلة (خطة NS) والخطة الجديدة المهيكلة (خطة S)، وعرضت المقارنة مع إطار العمل متعدد العروض في الجدولين 10-4 و10-5 على التوالي. كما يبيّن الجدول 10-6، على الرغم من أن عدد الأهداف الإجمالي أكبر في الخطة S إلا أن معدل تعقد التفسير أقل. وهذا يعود إلى عدد المقاييس القليل لكل هدف متحقق كنتيجة لتطبيق إطار العمل متعدد العروض على الخطة NS. عدد التبعيات وكثافة التبعيات يساوي صفراً. لذا، يكون الاستقصاء بأثر رجعي هو أول عملية تحقق إيجابية من صحة إطار العمل متعدد العروض.

الجدول (10 ـ 4)

إسناد ترافقي للمخطط NS (NS - Plan)

	العملية				المنتج			
الأهداف	تصميم سيناريو اختبار	اختبار النظام	تصحيح العيوب	التنفيذ	اختبار النظام	تنفيذ النظام	إدارة المشروع	كفاءة الاستثمار
هـ 1		x	x	x	x			
هـ 2							x	
هـ 3					x	x		
هـ 4	x	x				x		
هـ 5					x	x		
هـ 6		x					x	x
هـ 7					x		x	x
هـ 8			x		x			

الجدول (10 ـ 5)

الإسناد الترافقي للخطة S

	العملية				المنتج			
الأهداف	تصميم سيناريو اختبار	اختبار النظام	تصحيح العيوب	التنفيذ	اختبار المنتج	تنفيذ النظام	إدارة المشروع	كفاءة الاستثمار
هـ تصميم سيناريو اختبار	X							
هـ اختبار النظام		X						

يتبـــع

تابـــع

هـ تصحيح العيوب			X					
هـ التنفيذ، 1				X				
هـ التنفيذ، 2				X				
هـ نظام اختبار					X			
هـ تنفيذ النظام، 1						X		
هـ تنفيذ النظام، 2						X		
هـ إدارة المشروع							X	
هـ كفاءة الاستثمار، 1								X
هـ كفاءة الاستثمار، 2								X

يتم تنفيذ الاستقصاء عن المشروع في بيئة المشروع نفسه للتأكد من عمليات تطوير أو إنتاج أو تقييم التطبيقات الجديدة أو العمليات أو التقنيات أو الأدوات. وهذا مكلف ويحتمل مخاطر كبيرة نظراً إلى أنها تتطلب مطوّرين محترفين عليهم العمل حسبما يقتضي تصميم الاستقصاء؛ فهو يتطلب أن يتم تنفيذ تصميم الاستقصاء مع مراعاة الشروط وجمع المقاييس التي يتطلبها التصميم. وبالتالي، فإن الوقت المطلوب لتنفيذ المشروع يكون أكبر مما هو متوقع غالباً. أيضاً، قد يحدث أداء أسوأ من الأداء المتوقع من دون إجراء معالجة. خلاصة ذلك، إن هذا النوع من الدراسات مكلف ومحفوف بالمخاطر بالنسبة إلى جودة المنتج وأوقات تنفيذ المشروع التجريبي. بناءً على ظروف التنفيذ، يمكن تصنيف استقصاء المشروع كما يظهر في الجدول 10-7.

الجدول (10 ـ 6)

المتغير التابع

البيانات المجمعة	الخطة NS	الخطة S
عدد الأهداف	8	11
عدد القياسات	168	168
عدد المقاييس لكل هدف (معدل)	32.75	20.18
عدد التبعيات	24	0
كثافة التبعيات	0.27	0

الجدول (10 ـ 7)
تصنيف استقصاءات المشروع

	المتغيرات التابعة في الاستقصاء	
عدد المشاريع اللازمة لتنفيذ الاستقصاء	معرف مسبقاً	معرف في أثناء الاستقصاء
واحد	حالة الاستقصاء	استقصاء استكشافي
أكثر من واحد	استقصاء ميداني	استقصاء استكشافي ميداني

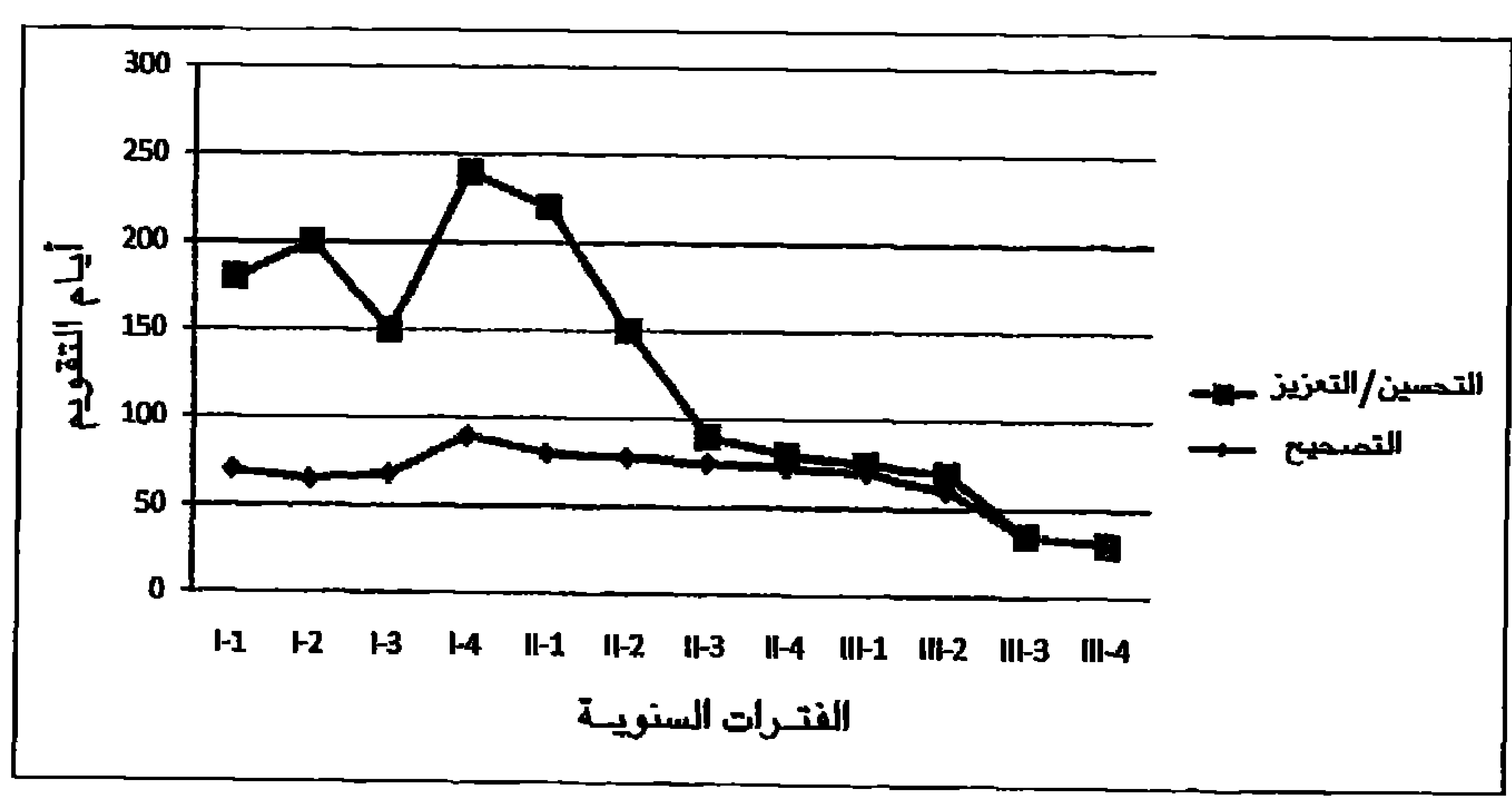
للتحقق من فعالية المعالجة، يمكن مقارنة مشروعين: أحدهما يطبّق المعالجة والآخر مشروع تحكم ينفّذ في ظروف عادية من دون تطبيق أي معالجة. نادراً ما تكون الشركة على استعداد لتنفيذ مشروعين بنطاق العمل نفسه، بمراعاة أن أحدهما ينفّذ لأغراض تجريبية فحسب من دون اعتبار الأغراض الإنتاجية. إضافة إلى ذلك، من الصعوبة بمكان إنتاج السياق نفسه لمشروعين، أحدهما ينفّذ مع معالجة والآخر ينفذ ضمن شروط طبيعية. بهذه الطريقة. لا تقارن البيانات التي جمعت في المشروع الأول بالبيانات التي جمعت في المشروع الثاني. غالباً ما تستخدم السلاسل التاريخية كـ O1، O2، . . .، X1، O1، O2، . . .، On، X2، . . . حيث Oi المشاهدات للبيانات المجمعة وXi المعالجات. أما الزمن بين مشاهدتين متتابعتين فتكون ثابتة ويشار لها بزمن العيّنة[24]. أما الأداة الإحصائية المستخدمة في هذه الحالة فتتضمن أيضاً تحليل للسلاسل التاريخية. يجب التحقق من أن اتجاه المشاهدات قبل المعالجة يختلف عنه بعد المعالجة، وأن الفرق بين الاتجاهات يعزى إلى المعالجة.

مثال على ذلك، سنأخذ في الاعتبار خبرة الشركة التي كانت تواجه مشكلة في تقليل طلبات صيانة تطبيقات الأعمال والحد من فترات الانتظار بهدف تحسين مستوى الرضا. تفاصيل هذا المثال موجودة في المرجع[40].

النظرية المفترضة هي أن تعميم قدرات التطبيق قد تلبّي احتياجات طيف واسع من المستخدمين. بذا، عند إضافة عملاء جدد أو أن استخدام التطبيق يتغير مع الزمن بالنسبة إلى العملاء الموجودين، تكون معظم طلباتهم متنبأ بها من التطبيق نفسه، تبعاً لذلك، يقل عدد طلبات الصيانة. إذا كان بالإمكان تكييف التطبيق مع متطلبات العميل من دون تعديل البرمجية، فإنه يمكن إنجاز ذلك في فترة قصيرة. إذا كان للتطبيق هيكلية قائمة على الوحدات، ومستوى

التعقيد المنخفض لم يتأثر سلبياً بطلبات الصيانة السابقة، فإنه يتوقع عدد مرات الصيانة لن تزيد مسألة الوقت سوءاً.

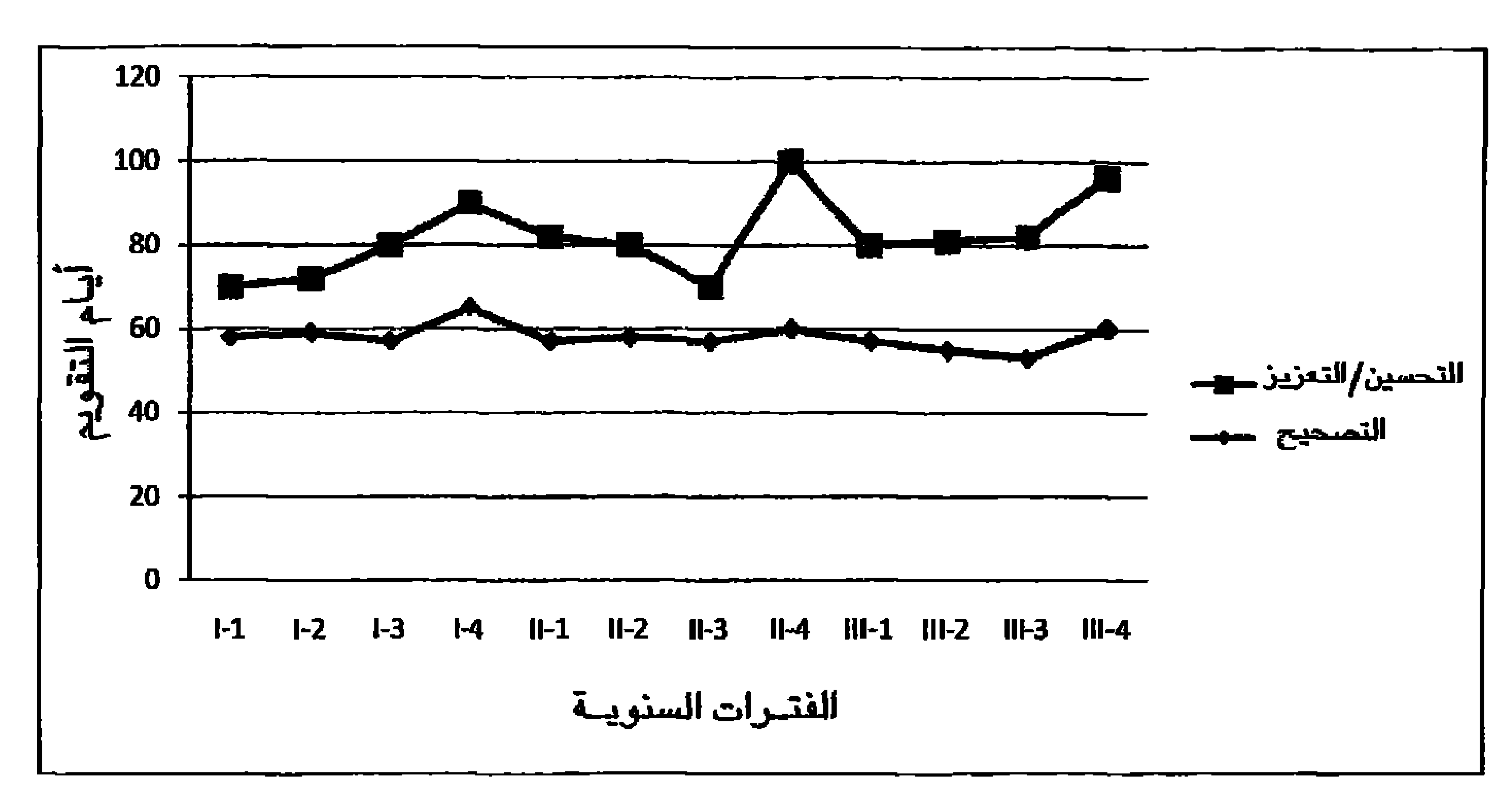
لتحقيق هذه النظرية، قامت الشركة لنموذج تحليل جديد للتطبيق بالرجوع إلى ملف مستخدم نظري يعمم طلبات المستخدم الفعلية؛ يطلب كل عميل مجموعة من الإمكانيات. إضافة إلى ذلك، يتم تطوير أسلوب جديد للتصميم مشترك مع الوحدات والأنماط المعلمية قادر على ضمان إعادة استخدام البرمجية، ودرجة تعقيد الهيكلية المنخفض وتخصصية سلوك التطبيق عن طريق تخصيص قيم لمجموعة من المعاملات. أخيراً، تم تحديد عملية صيانة تعرف بالتحسين التكراري قادرة على الحفاظ على وحدات البرمجية وإمكانية تتبعها. للتحقق من صحة هذه التقنيات، تم تطبيقها على بعض التطبيقات المستخدمة في القطاع المصرفي، لكنها لا تستخدم في الإدارة العامة الحالية أو في القطاع الصحي. في القطاعين الأخيرين، تبقى التقنيات نفسها من دون تغيير. تم جمع البيانات على أساس الفصل؛ بدأ تأثير المعالجة في القطاع المصرفي خلال عام؛ في العام الثاني، ظهر تأثير التطبيقات المتوارثة في القطاع نفسه، التي تمّ تجديدها حسب الأنماط الجديدة والهيكلية القائمة على الوحدات المستخدمة في التطبيقات الجديدة. يبيّن الشكلان 10-2 و10-3 طلبات الصيانة في القطاع الذي يتم تطبيق المعالجة عليه، وفي القطاعين الأخريين اللذين لم يعالجا. يبيّن الشكلان 10-4 و10-5 معدل زمن الانتظار لتنفيذ الطلبات المصنفة من قبل التطبيقات في كل القطاعات.



الشكل (10 ــ 2): اتجاهات صيانة الاعتراضات التي تتطلبها البنوك

يبيّن الشكل 10 ـ 2 أن المعالجات ذات تأثير وثيق الصلة بتحسين وتعزيز البرمجية. في الواقع، ينخفض الاتجاه على نحو ذي مغزى بعد المعالجة.

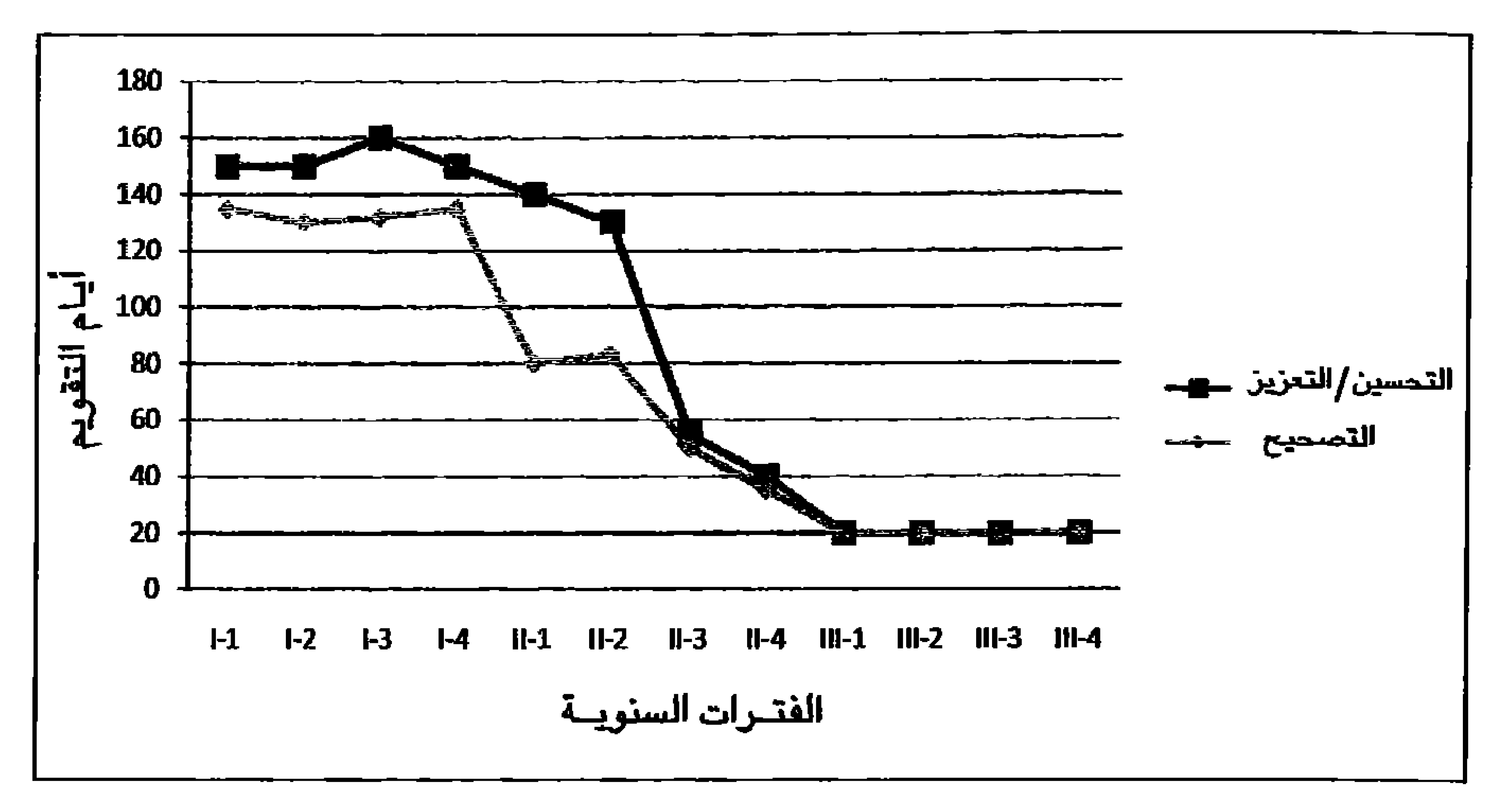
تم تأكيد هذه النتيجة في الشكل 10 ـ 3: في القطاعين اللذين لم يتعرضا للمعالجة، اتجاه عدد التعديلات كبير ويزيد مع الوقت. لا يوجد هناك تغييرات مهمة للصيانة التصحيحة؛ لذا، لا تؤثر هذه المعالجة إيجابياً في انتشار الخطأ في البرامج.



الشكل (10 ـ 3): اتجاهات صيانة الاعتراضات التي تتطلبها الإدارة العامة المحلية والمنظمات في القطاع الصحي

يشير الشكل 10 ـ 4 إلى أن فترة الانتظار لتلبية طلب تقل بعد المعالجة؛ مرة أخرى، هذا الأمر مثبت عن طريق مراقبة أن الفترة تقل عند توسيع عملية وضع بنية تركيبية للبرامج لتشمل النظم المتوارثة.

من الأهمية بمكان ملاحظة أن الاتجاه في الشكل 10 ـ 4 يميل نحو خط المقاربة، في حين يميل في الشكل 10 ـ 5 إلى زيادة مطّردة. وهذا يشير إلى أن معالجة الاستقصاءات التجريبية تقلّل من تدهور حالة البرمجية. أخيراً، لاحظ أنه لإجراء التصحيح اللازم، يقل الاتجاه للتطبيقات التي أنتجت باتباع العمليات التي تكون عرضة للمعالجة. وهذا يعني أن المنتج البرمجي الناتج أكثر قابلية للصيانة.



الشكل (10 ـ 4): اتجاهات زمن الانتظار لصيانة الاعتراضات التي تتطلبها البنوك

يتم تنفيذ التجربة «تجربة/تحكم» في بيئتها (أي بوجود عوامل متحكم بها) لإحراز معرفة جديدة. على وجه الخصوص، يرغب الباحث في إثبات أن المعالجة التي خطط لها في التجربة لها علاقة سببية بالمتغيرات التابعة. تقسم الفرضية المفاهيمية للتحقق من صحة ذلك إلى فرضيتين فعالتين:

- فرضية العدم، H_0: تحدد ما يرغب الباحث في تحقيقه، وهو أنه ليس صحيحاً لعيّنة الدراسة. الهدف هنا هو رفض الفرضية مع أكبر دلالة ممكنة.
- الفرضية البديلة، H_1/H_a: تحدد التوكيد لصالح فرضية العدم المرفوضة.

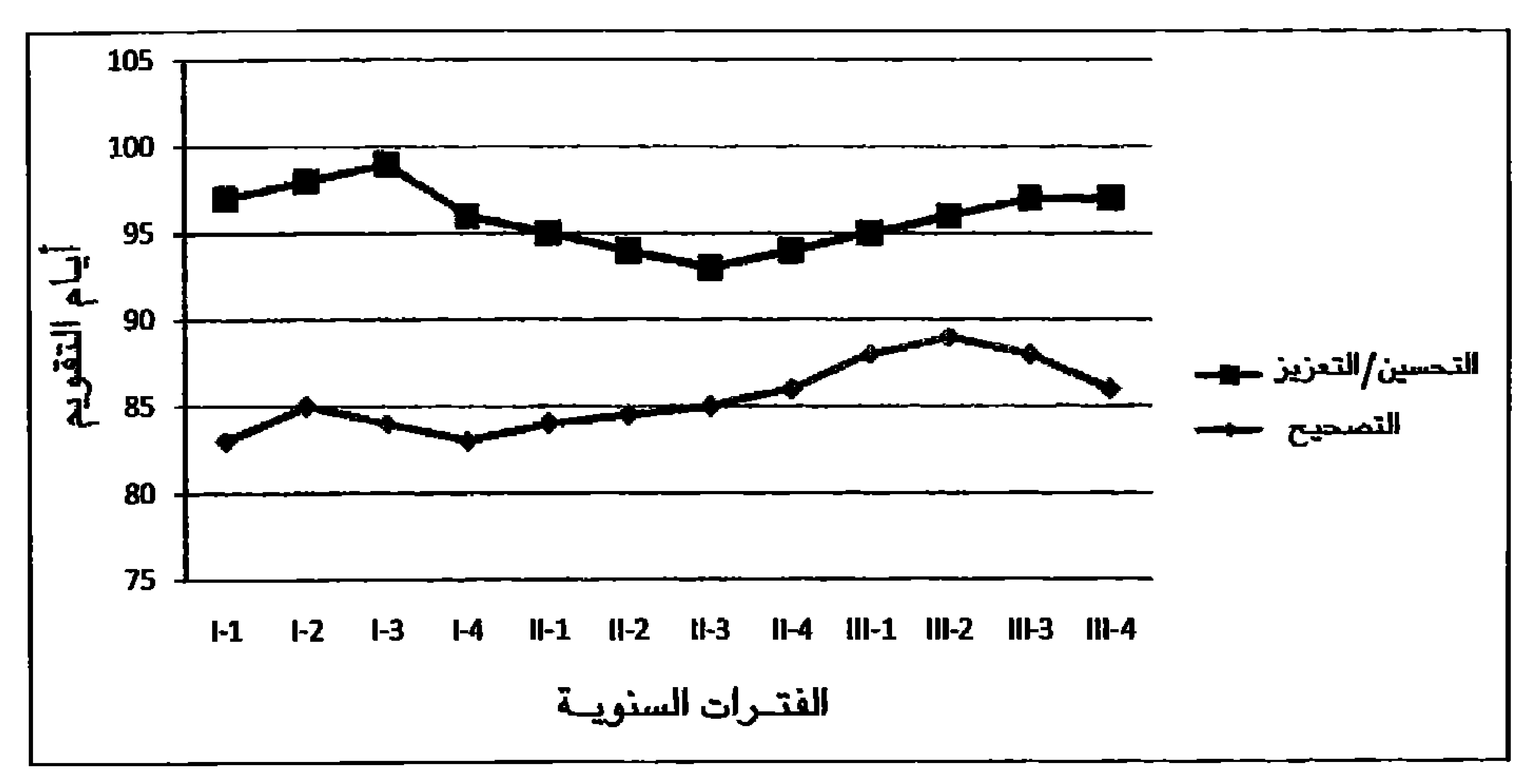
تتميز التجربة بالتصميم الذي يطبّق المعالجات على مواضيع الدراسة بهدف مراقبة التأثير. على سبيل المثال، إذا كان العامل 1 (لغة برمجة) بمعالجتين (لغة جافا ولغة سي) وعدد أفراد المجتمع التجريبي س، فإن كل موضوع يكون عرضة لكل معالجة بتنفيذين تجريبيين حسب الجدول 10-8.

إن تخصيص كل موضوع من مواضيع الدراسة يخضع لتوليفة من معالجتين عشوائيتين، بحيث يتوزع مجتمع الاستقصاء التجريبي بالتساوي بين المعالجتين.

عندما يؤدي عدد العوامل وعدد المستويات لكل عامل إلى عدد كبير من التوليفات، يتم تصميم المشروع التجريبي بتقنية التصميم العاملي المخفض.

يبيّن الجدول 10-9 مثالاً بوجود ثلاثة عوامل (طريقة الصيانة، واللغة ونوع الصيانة)، لكلّ منها مستويان محتملان هما على النحو: إصلاح سريع QF أو

تحسين تكراري IE، لغة جافا أو لغة + +C، صيانة تصحيحية Cor أو صيانة تطويرية Ev.



الشكل (10 ـ 5): اتجاهات زمن الانتظار لصيانة الاعتراضات التي تتطلبها الإدارة العامة المحلية والمنظمات في القطاع الصحي

الجدول (10 ـ 8)

مثال على الموضوعات التجريبية

التوزيع في استقصاء عامل واحد ـ عمليتي معالجة

الموضوعات التجريبية	التنفيذ الأول	التنفيذ الثاني
S_1	C+ +	Java
S_2	Java	C+ +
S_3	Java	C+ +
S_4	C+ +	Java
...	...	...
S_n	Java	C+ +

هناك عدد من الاختبارات الإحصائية المختلفة الموصوفة في الأدبيات التي يمكن استخدامها لتقييم نتائج التجربة. يتضمن اختبار الفرضية مخاطر مختلفة يشار إليها بالخطأ من النوع الأول والخطأ من النوع الثاني. يحدث الخطأ من النوع الأول عندما يشير الاختبار الإحصائي إلى وجود علاقة نمطية حتى لو كان

لم يكن هناك نمط فعلي: رفض فرضية العدم H_0 عندما تكون فرضية العدم H_0 صحيحة. يحدث الخطأ من النوع الثاني عندما لا يشير الاختبار الإحصائي إلى علاقة نمطية حتى لو كان هناك نمط فعلي: عدم رفض فرضية العدم H_0 عندما تكون فرضية العدم H_0 خاطئة. للإكمال، تكون قوة الاختبار الإحصائي هي احتمالية أن يرفض الاختبار H_0 عندما تكون خاطئة. كما إنها متممة 1 لاحتمالية الخطأ من النوع الثاني.

تقسم جميع الاختبارات الإحصائية الفضاء العيني (فضاء المعاينة أو الحدث) إلى جزءين متممين: مساحة لقبول فرضية العدم H_0 ومساحة لرفض فرضية العدم H_0. في تجربة، إذا كانت القيمة الإحصائية لاختبار تقع في المساحة الأولى، يتم قبول فرضية العدم؛ أما إذا وقعت في المساحة الثانية، فيتم رفض فرضية العدم لصالح قبول الفرضيات البديلة. تقسم جميع الاختبارات الإحصائية المساحة بحيث تكون المساحة الأولى أكبر من المساحة الثانية، وذلك لتجنب الاستنتاجات الخاطئة ـ أي قبول الفرضية البديلة H_1 عندما لا تكون صحيحة بالفعل[44, 25, 24]. بمعنى آخر، عند تنفيذ اختبار ما، يكون من الممكن في العديد من الحالات حساب أقل أهمية ممكنة لرفض فرضية العدم: القيمة p عندما تكون القيمة p أكبر من 0.05، يتم رفض H_0؛ كلما كانت قيمة p أقل، زادت أهمية الرفض.

الجدول (10 ـ 9)

مثال على الموضوعات التجريبية – التوزيع في استقصاء ثلاثة عوامل (كل عامل بمعالجتين)

موضوع التجربة	التنفيذ الأول	التنفيذ الثاني
S_1	I.E., Java, Cor	Q.F.,C++, EV
S_2	Q.F.,J, Cor	I.E.; C++,Ev
S_3	I.E.; C++,Ev	Q.F.,J, Cor
S_4	Q.F.,C++, Cor	I.E., Java, EV
...	...	...
S_n	Q.F., Java, Cor	I.E.; C++,Ev

يوفّر الإحصاء أنواعاً عديدة من الاختبارات التي يمكن تصنيفها إلى معلمية (parametric) وغير معلمية (nonparametric).

ترتكز الاختبارات المعلمية على النموذج الذي يتضمن توزيعاً محدداً. أما الاختبارات غير المعلمية فلا تأخذ نفس نوع الافتراضات في ما يتعلق بتوزيع العوامل. يلقي الجدول 10 ـ 10 نظرة عامة عن الاختبارات المعلمية أو غير المعلمية لتصاميم مختلفة للتجربة؛ لمزيد من التفاصيل، راجع المراجع[25, 24] و[44].

كمثال، نقدم لمحة عامة عن تجربة مسيطر عليها من حيث شمولية وفعالية تصميم خطة القياس المصممة باستخدام إطار العمل متعدد العروض MF الموضح مسبقاً في هذا الفصل. لمزيد من المعلومات، يمكن للقراء المهتمين الرجوع إلى المرجع[7]. سياق التجربة هو المادة التعليمية «هندسة البرمجيات» في جامعة باري. تمّت صياغة هدفين من أهداف البحث لتقييم كفاءة وشمولية الخطة S مقارنة بالخطة NS:

H_0^{RG1}: لا يوجد فرق في الكفاءة (الجهد المبذول في التفسير) بين تفسيرات الخطة S والخطة NS

H_1^{RG1}: هناك فرق في الفعالية بين الخطة S والخطة NS.

H_0^{RG2}: هناك فرق في الشمولية (التفسيرات معرضة للخطأ) بين الخطة S والخطة NS.

H_1^{RG2}: لا يوجد فرق في الشمولية بين الخطة S والخطة NS.

المتغيرات التابعة هي: الجهد المبذول، أي الفرق بين زمن بدء وزمن انتهاء تفسير خطة القياس كاملة (الخطة S أو الخطة NS)؛ التعرض للخطأ، أي عدد الاستنتاجات غير الصحيحة في تفسير أهداف خطة القياس (الخطة S أو الخطة NS). أما مجتمع الدراسة فكان طلاب الدراسات العليا المقرر عليهم مادة هندسة البرمجيات. أخذت عيّنة الدراسة من 34 مجتمعاً تجريبياً بشكل عشوائي، وُزِّعوا على مجموعتين متساويتين من 17 شخصاً، عرّفت كـ «مجموعة أ» و«مجموعة ب». تم تدريب جميع الطلاب بنفس الطريقة، وبناءً على ذلك، كان لديهم نفس الخبرة والمعرفة. كان تصميم التجربة يتضمن عاملاً واحداً (نموذج جودة الخطة) ومعالجتين (الخطة NS والخطة S)، ونظّمت في تنفيذين تجريبيين (التنفيذ 1 والتنفيذ 2)، وأجريت على مدار يومين، يبيّن الجدول 10 ـ 11 تصميم التجربة. أما الجدولان 10-12 و10-13 فيلخصان النتائج التي تحقق صحة فعالية خطط القياس باستخدام إطار العمل متعدد العروض MF.

الجدول (10 ـ 10)

مقدمة عن الاختبارات الإحصائية

التصميم	معلمية	غير معلمية
عامل واحد، معالجة واحدة		اختبار Chi-2 ذو حدين
عامل واحد، معالجتان، تصميم عشوائي تماماً	t-Test	Mann-Whitney Chi-2
عامل واحد، أكثر من عمليتي معالجة	ANOVA	Kruskal-Waills Chi-2
أكثر من عامل واحد	ANOVA	

الجدول (10 ـ 11)

تصميم التجربة

المجموعة/ التنفيذ	التنفيذ الأول	التنفيذ الثاني
المجموعة أ	S-GQM	NS-GQM
المجموعة ب	NS-GQM	S-GQM

10 ـ 2 ـ 3 مخاطر ومهددات الاستقصاء عن الصلاحية

بالرجوع إلى الشكل 10 ـ 1 من النظرية إلى المشاهدة أو العكس، يتوجب على الباحث القيام بالعديد من الخطوات، خطوة لكلّ سهم؛ في كل خطوة، ثمة مخاطر تتعلق بنتائج الاستقصاء عن الصلاحية. كما في المرجع[44]، تم تقدير بعض التفاصيل لقائمة من المخاطر الأساسية. يمكن الإطلاع على المزيد في المرجع[16].

يشير الاستقصاء عن صلاحية الاستنتاجات إلى العلاقة بين المعالجة والنتائج. المخاطرة المحتملة هنا هي عدم قيام الباحث برسم الاستنتاج الصحيح عن العلاقات بين المعالجة ونتيجة التجربة. في ما يأتي بعض المخاطر التي تنتمي إلى هذا الصف:

- قوة إحصائية ضعيفة: قد يكشف الاختبار عن نمط في البيانات غير الصحيحة.
- افتراض منتهك للاختبار الإحصائي: استخدام الاختبار الإحصائي على البيانات التي تفتقر إلى المميزات التي تضمن التنفيذ الملائم.

- موثوقية المقاييس: إن استخدام أساليب قياس سيئة أو نموذج قياس سيئ قد ينتج بيانات سيئة.
- موثوقية تنفيذ المعالجة: لا يكون تنفيذ المعالجة نفسه بين أشخاص مختلفين أو أحداث مختلفة.
- عناصر عشوائية في الإعداد التجريبي: العناصر الخارجية (كالأحداث أو الظروف) يمكن أن تسبب تشويشاً للنتائج.
- عشوائية عدم تجانس المواد الدراسية: تعتمد النتائج على الفروقات الفردية أكثر من اعتمادها على المعالجات.

الجدول (10 ـ 12)

قيم المستوى p في اختبار Mann-Whitney للجهد

تنفيذ التجربة	القيمة p للجهد	النتائج
التنفيذ 1 (المجموعة أ مقابل المجموعة ب)	0.00001	رفض H_0^{RG1} وقبول H_1^{RG1}
التنفيذ 2 (المجموعة أ مقابل المجموعة ب)	0.00002	رفض H_0^{RG1} وقبول H_1^{RG1}

الجدول (10 ـ 13)

قيم المستوى p في اختبار Mann-Whitney للتعرض للخطأ

تنفيذ التجربة	القيمة p للتعرض للخطأ	النتائج
التنفيذ 1 (المجموعة أ مقابل المجموعة ب)	0.00721	رفض H_0^{RG2} وقبول H_1^{RG2}
التنفيذ 2 (المجموعة أ مقابل المجموعة ب)	0.04938	رفض H_0^{RG2} وقبول H_1^{RG2}

تعنى الصلاحية الداخلية أيضاً بالعلاقة بين المعالجة والنتائج. قد يكون هناك مخاطرة من أن النتائج قد تشير إلى علاقة عرضية على الرغم من عدم وجود علاقة. في هذه الحالة، من الأفضل تسمية المخاطرة بالتهديد. من الناحية التقنية، التهديد هو حدث يتداخل مع تأثيرات المعالجة وينتج علاقة سببية (سبب ـ تأثير) غير موجودة فعلياً. في ما يأتي بعض التفاصيل عن التهديدات التي تنتمي إلى هذه الفئة:

- قد يحدث التاريخ عند تطبيق العديد من المعالجات على العيّنات نفسها

لأن سلوكهم يتأثر في أثناء المعالجة بتأثير المعالجات السابقة أو بتوليفة من المعالجات.

- قد يحدث النضوج عندما يستطيع الفرد التجريبي تعديل سلوكه بمرور الوقت. وقد ينتج تعديل السلوك من التعب أو الملل أو التعلم. إن إدارة هذه المخاطرة معقدة لأن أفراد العيّنة في التجربة ينضجون بسرعات مختلفة.
- قد يتم التحقق من الاختبار عند الانتهاء من إجراء العديد من الاختبارات على الأفراد نفسهم، لأن افراد العيّنة يغيرون سلوكهم لأنهم يعرفون كيف يتم تنفيذ الاختبار. وهذا نوع من النضج.
- أجهزة وأدوات الاختبار هي ما قد يؤثر في التسليمات والأدوات المستخدمة في أثناء الاستقصاء.
- الاختيار هو تأثير التباين الطبيعي في الأداء البشري وكيف يمكن أن تتأثر النتائج باختيار الأفراد. قد يكون اختيار متطوعين أو اختيار أفضل عشر أفراد من تجربة سابقة أو اختيار أفضل خبير في موضوع ما أمراً ينطوي على مخاطرة.
- قد يحدث فناء عندما يتسرب بعض الأفراد من التجربة لأنهم لا يمثلون العيّنة التجريبية.
- قد يحدث إسهاب أو تقليد عندما تتعلم مجموعة التحكم أو تقلد سلوك مجموعة الدراسة.
- قد تحدث منافسة تعويضية عندما ينفذ الفرد معالجات لا يكون راغباً بها لأن لديه حوافز ودوافع لتقليل أو عكس النتائج المتوقعة.
- الإرباك الذي يسبب الاستياء بعكس النقطة السابقة. قد يتخلى الفرد الذي تُجرى عليه معالجات لا يكون راغباً بها عن المهمة ولا ينجز عمله بالكفاءة نفسها التي يعمل بها عادة.

تُعنى صلاحية التركيب بالعلاقة بين النظرية والمشاهدة. تشير المخاطرة إلى المدى الذي يعكس فهي إعداد التجربة التركيب الذي سيتم اختباره فعلياً. المخاطر التي قد تنجم هي: المعالجة لا تعكس تركيب السبب جيداً، أو أن النتائج لا تعكس تأثير التركيب جيداً. قد ينتج ذلك من نقاط ضعف التحليل الإحصائي الذي يؤدي إلى علاقة سببية (السبب ـ التأثير) بين المعالجة والنتائج.

في ما يأتي بعض التفاصيل عن المهددات التي تنتمي إلى هذه الفئة:

- قد ينتج عدم كفاية التحليل المنطقي اللازم للتشغيل لتركيب الدراسة عندما لا تكون التراكيب محددة بما فيه الكفاية لأنها تترجم على أنها مقاييس ومعالجات غير فعالة.

- قد ينتج الانحياز الأحادي عندما يكون في الدراسة متغير مستقل واحد فقط أو حالة واحدة للدراسة أو فرد واحد للتنفيذ أو معالجة واحدة أو مقياس واحد. في هذه الحالة، قد يكون الاستقصاء بدون مستوى التمثيل للتركيب لأنه لا يمكن إجراء تحقق متبادل لمتغير مستقل أو حالة أو فرد أو معالجة أو قياس مع الآخر.

- عوامل مربكة في ما يتعلق بمستويات العوامل التي قد تكون نتجت عندما لا يكون للعامل (أي التجربة) عدد محدد من المستويات لأن تأثير المعالجات قد يكشف عنها من قبل مستويات مختلفة لذلك العامل. على سبيل المثال، إن الفرق النسبي في التجربة هو بين سنتين إلى ثلاث سنوات، أو بين ثلاث سنوات إلى ست سنوات؛ في حين أن فترة من أربع سنوات إلى خمس سنوات من عمر التجربة ليس له تأثير، لذا فإن مستويات التجربة يجب أن تكون 2، 3، 6 أو سنوات أكثر. سيكون من الخطأ تحديد مستويين: أقل من أو يساوي 2، أو أكثر من 3.

تعنى الصلاحية الخارجية بالتعميم. المخاطر المحتملة هي: خطأ اختيار أفراد التجربة أو خطأ في تحديد البيئة والأداء أو توقيت خاطئ بحيث تتأثر النتائج بخصائص التجربة الأصلية التي تم تغييرها. في ما يلي بعض التفاصيل عن المهددات التي تنتمي إلى هذه الفئة:

- قد يتم إنشاء مجتمع الدراسة عندما يكون الأفراد الذين اختيروا لإجراء التجربة لا يمثلون مجتمع الدراسة. وهكذا فإنه من غير الممكن تعميم نتائج الاستقصاء.

- قد يتم إنشاء الإعدادات عندما لا تكون الإعدادات أو المادة المستخدمة للنسخ المماثلة ممثلة للسياق، ما يجعل تعميم النتائج أمر غير منطقي.

- قد يتم إنشاء الفترة عندما تكون نتائج الدراسة متحيّزة لفترة التشغيل المحددة.

10 ـ 2 ـ 4 إرشادات توجيهية للتجربة

عندما يكون تصميم التجربة ضعيفاً، لا تكون الاستنتاجات موثوقة، وغالباً ما لا تكون ملخصة من البيانات التجريبية. واجه العديد من الباحثين هذه المشكلة عندما هدفوا إلى وضع إرشادات لتحسين تصميم وتصميم التجربة[1-3، 13، 18، 19، 23، 28، 34، 37، 43]. من المفيد تحليل بعض المظاهر الحاسمة للتجربة وإعطاء بعض الاقتراحات لكلّ منها. أما أهم مصادر الملاحظات التالية فهي المرجع[28] وخبرة المؤلف.

يجب أن يتم تحديد سياق التجربة مقدماً، على وجه الخصوص. يجب تحديد الكوائن الضالعة في التجربة والخصائص والمقاييس اللازمة للتنفيذ لإضفاء الطابع الرسمي على توصيف السياق. على سبيل المثال، العوامل التي يمكن توصيفها هي: البيئة التي ينفّذ فيها المشروع (في المنزل أو شركة برمجيات مستقلة أو مجموعة عمل في الجامعة وغير ذلك)؛ وكفاءات وخبرات الموظفين الضالعين في المشروع؛ ونوع البرمجية المستخدمة (أدوات تصميم أو معالجات أو أدوات اختبار، وغير ذلك)، وعملية البرمجة المستخدمة (عملية تطوير معيارية لشركة ما، أو عملية ضمان الجودة أو عملية إعداد وتهيئة، وغير ذلك). لسوء الحظ، يصعب تعريف العديد من متغيرات السياق في عمليات البرمجيات. يجب إجراء الكثير من الأبحاث لتحسين قدراتنا التعريفية. إحدى الأمثلة ذات العلاقة في هذا السياق، ما أورده Kichenham وآخرون[27] الذي يبيّن الخطوط العريضة لتعريف عوامل توصيف السياق لصيانة برمجية.

يجب أن يتم تحديد فرضية البحث/الاستقصاء التي سيتم اختبارها بوضوح مقدماً. بصورة أدق، يجب أن تكون الفرضية مرتبطة بالنظرية الأساسية التي يجب التحقق من صحتها. عند عدم وجود هذا التتبع، فإن نتيجة البحث لن تسهم في توسيع نطاق المعرفة الحالي. في هندسة البرمجيات، غالباً ما تكون نتائج التجارب التي تهدف إلى إثبات الفرضيات نفسها متناقضة. إذا كانت الفرضية والنظرية غير قابلتين للتتبع، فإنه يصعب تفسير النتائج وتناقضها، كما يصعب استخدامها. الفرضيات التي نشير إليها هي تأكيدات (تصريحات) يجب إثباتها من خلال تجربة، التي أشرنا لها مسبقاً بفرضية العدم/الفرضية البديلة H_0/H_1. لتجنب الخلط، يستخدم بعض المؤلفين تعبيرات أخرى كاستفسارات البحث/الاستقصاء أو أهداف البحث/الاستقصاء بدلاً من فرضيات. على سبيل

المثال، التجربة التي تهدف إلى إثبات التوكيد الآتي ليس لها معنى: العدد السيكلوماتي لوحدة ما يرتبط بعدد الأخطاء الموجودة فيها. تشير بعض التجارب إلى أن عدد الأخطاء يقل كلما زاد العدد السيكلوماتي، وأن المتغيرين غير مرتبطين بعلاقة تبادلية. في حين أظهرت تجارب أخرى كيف أن عدد الأخطاء يقل بارتباط وثيق بين المتغيرين عندما يقل العدد السيكلوماتي. لتفسير هذه النتيجة الأخيرة، تم افتراض أنه عندما يكون العدد السيكلوماتي كبيراً، فإن المطوّرين يولون اهتماماً أكبر بالبرمجة، وبناء على ذلك تكون الأخطاء الناتجة أقل. هذه المعرفة غير مدعومة بدليل تجريبي في تطوير هندسة البرمجيات. يمكن تقديم تفسير أفضل إذا رجعنا إلى نظرية بارنز؛ إن الارتباط بين درجة التعقيد السيكلوماتي(*) لوحدة ما والأخطاء المعرفة يجب أن يرتبط بكفاءة الجهد الذي يجب أن يتحمله المطوّر لفهم محتويات الوحدة وسلوكها.

يجب أن يتم تحديد تصميم التجربة بحيث يكون أفراد العيّنة التجريبية وأدوات التجربة ممثلين لسياق الاستقصاء. إضافة إلى ذلك، يجب أن يحدد مقدماً كيف سيتم اختيار أفراد العيّنة التجريبية وكيف سيخضعون لكل معالجة. يجب أن يكون المخطط الناتج من التصميم قادراً على الحد ما أمكن من الأسباب التي قد تؤدي إلى وجود تهديدات لنتائج التجربة أو التوقع بكيفية قياس تأثيرها في النتائج. عند تحديد التصميم التجريبي، يجب أن يتم توضيح إذا ما وجدت فروقات بين عيّنات التجربة والوحدة التجريبية. فإن وجدت فروقات، وَجَبَ تتبع الأفراد والوحدة مع بعضهما البعض. غالباً ما يخلط القائمون على التجربة بينهما، لذا، إذا كان لابدّ من توزيع استبيانات، يجب أن يشير التصميم التجريبي إلى أن أفراد العيّنة التجريبية هم أولئك الذين سيجيبون عن الاستبيانات، في حين أن الوحدات التجريبية هي المؤسسات التي تتم فيها المقابلات لجمع بيانات التجربة.

تتضمن كل وحدة العديد من العيّنات التجريبية. عند تصميم تجربة، من المفيد اعتماد مخططات معروفة في الأدبيات التي اعتمدت ودعمت في الأدلة. وهذا سيجنب التجربة المخاطر عديمة الفائدة والحد من التهديدات التي تؤثر

(*) مقياس أو مؤشر مكاب (McCabe Complexity) أو ما يعرف بدرجة التعقيد السيكلوماتي (Cyclomatic Complexity) هو مقياس أو قيمة يمكن من خلالها قياس درجة تعقيد برنامج أو خوارزمية معينة. تم تطوير هذا المقياس من قبل توماس ج. ماكابي عام 1976 (المترجم).

في صلاحية النتائج. إضافة إلى ذلك، يجب أن يتم توقع مستويات التحكم الملائمة لتجنب أن تؤثر توقعات المشاركين والباحث الذي يجري الاختبار في النتائج بطريقة أو بأخرى. في التجربة، يجب أن تكون المتغيرات المستقلة معرّفة بوضوح وذات دوافع مع كيفية تحليل البيانات المرتبطة. إضافة إلى ذلك، يجب أن يكون نطاق المتغيرات المستقلة والتحليلات الإحصائية مناسبة، وتتجنب التعارضات المفاهيمية؛ وبخلاف ذلك، قد يؤدي تفسير النتائج إلى التباس. أخيراً، عند تصميم تجربة، من الأفضل تجنب التحكم بسلوك أفراد عيّنة التجربة. بمعنى أدق، يجب أن لا يشعر أفراد العيّنة أنهم مراقبون أو مسيطر عليهم، خصوصاً في مشروع الاستقصاء. في هذه الحالة، قد يتأثر الأفراد بسلوكهم عندما يعرفون أنه مسيطر عليهم، وقد يتصرفون بطريقة مختلفة عن تصرفهم في الظروف الطبيعية. لذا، على سبيل المثال، من الأهمية بمكان عدم تعديل المقاييس المألوفة وجمعها على أساس منتظم. ينصح باستخدام المقاييس التي يمكن جمعها على المنتجات من دون معرفة المطوّرين بها، أو استخدامها لدعم العمليات القادرة على توفير معلومات حول العمليات المختبرة[(8)]. لكن، ليست تلك هي الحالة التي تأخذ في الاعتبار النتائج التجريبية كتقنية لتقييم أداء المطوّر.

يجب أن تكون البيانات التي يجب جمعها موضوعية ومحددة. أما البيانات غير الموضوعية فيجب أن يتم تحديدها بدقة شديدة وذلك لتقليل درجات الحرية لدى من يجري القياس. إضافة إلى ذلك، يجب أن يتم تحديد وسائل الجمع والتحكم بدقة أيضاً.

يجب أن يحدد عرض وتفسير النتائج الكيفية التي رسم بها الباحث استنتاجاته والسماح للقراء المهتمين التحقق منها. يوضح المرجع[28] إرشادات لإنتاج تقارير عن نتائج التجربة ويدرج العديد من المراجع التي تفصل القضايا التي تواجه عملية نشر النتائج. حالما يتم بيان استنتاجات التجربة، يجب أن يتم توفير وثائق مناسبة تصف التجربة بحيث تكون قابلة لنقلها إلى باحثين آخرين. لهذا الغرض، تقدم أدبيات الاستقصاء اقتراحات متنوعة[(30, 26)]. في ما يلي بعض الإرشادات الموجزة عن إنتاج الوثائق التي تكوّن حزمة الاستقصاء:

- الوصف التفصيلي للمعالجات بما في ذلك كافة أنواع التعليمات والأدوات التي تدعم التنفيذ.

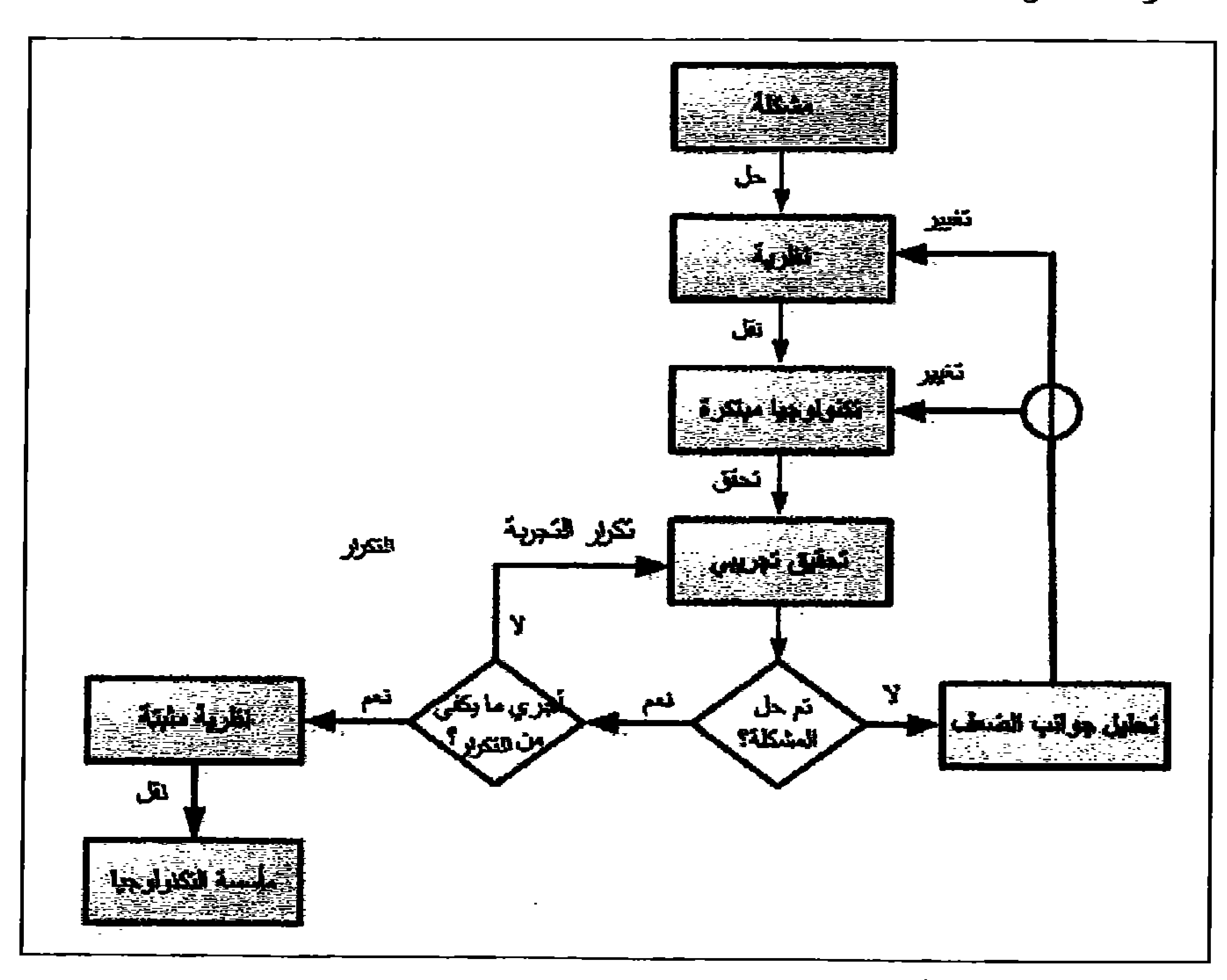
- العناصر التجريبية التي تطبق عليها المعالجة؛ على سبيل المثال، الشيفرة البرمجية وتصميم البرمجية أو مواصفاتها.
- تعليمات لوصف السلوك الدقيق للجهات المعنية في الاستقصاء، بما في ذلك الباحثون.
- تفاصيل الدراسات التجريبية أو تجارب المحاولات.
- آليات لجمع البيانات التجريبية والتحكم بها، سواء كانت مؤتمتة أو يدوية.
- المواد التعليمية لإعداد أفراد المجتمع التجريبي للاستقصاء.
- البيانات الخام التي جمعت في التجربة الأصلية، عندما يكون ذلك ممكناً.
- الوصف والمقاييس وأساليب جمع كفاءات وقدرات أفراد المجتمع التجريبي وتحديد خصائص ذلك المجتمع والأفراد المختارين الذين يجب أن ينتموا له.
- قوة النسخ المتماثل، وفقاً للعيوب التي تضمنتها التجربة الأصلية والنسخ الأخرى، من حيث عدد وأنواع أفراد المجتمع التجريبي والتحكم بالبيانات أو التحليل الإحصائي لتنفيذ التجربة على البيانات التي تم جمعها.
- الأساس المنطقي لعملية صنع القرار في أثناء تصميم وتنفيذ التجربة، وذلك بهدف تجنب أن تغفل النسخ المتماثلة عن أي جوانب مهمة.
- تقييم تكلفة الاستقصاء.
- تحليل التكاليف ـ المنفعة في ما يتعلق بتطبيق النتائج وفقاً لقيمتها بالنسبة إلى المعنيين بالمشروع.

10 ـ 3 الدراسات التجريبية لعلم هندسة البرمجيات

10 ـ 3 ـ 1 لمحة عامة

يمثل الاستقصاء التجريبي مساهمة مهمة لتوطيد هندسة البرمجيات كعلم. في الواقع، تستخدم هندسة البرمجيات في كافة العلوم سواء كانت علوم طبيعية أو هندسية أو اجتماعية. تُعنى هندسة البرمجيات[33] بالتطوير من المشكلة المشاهدة

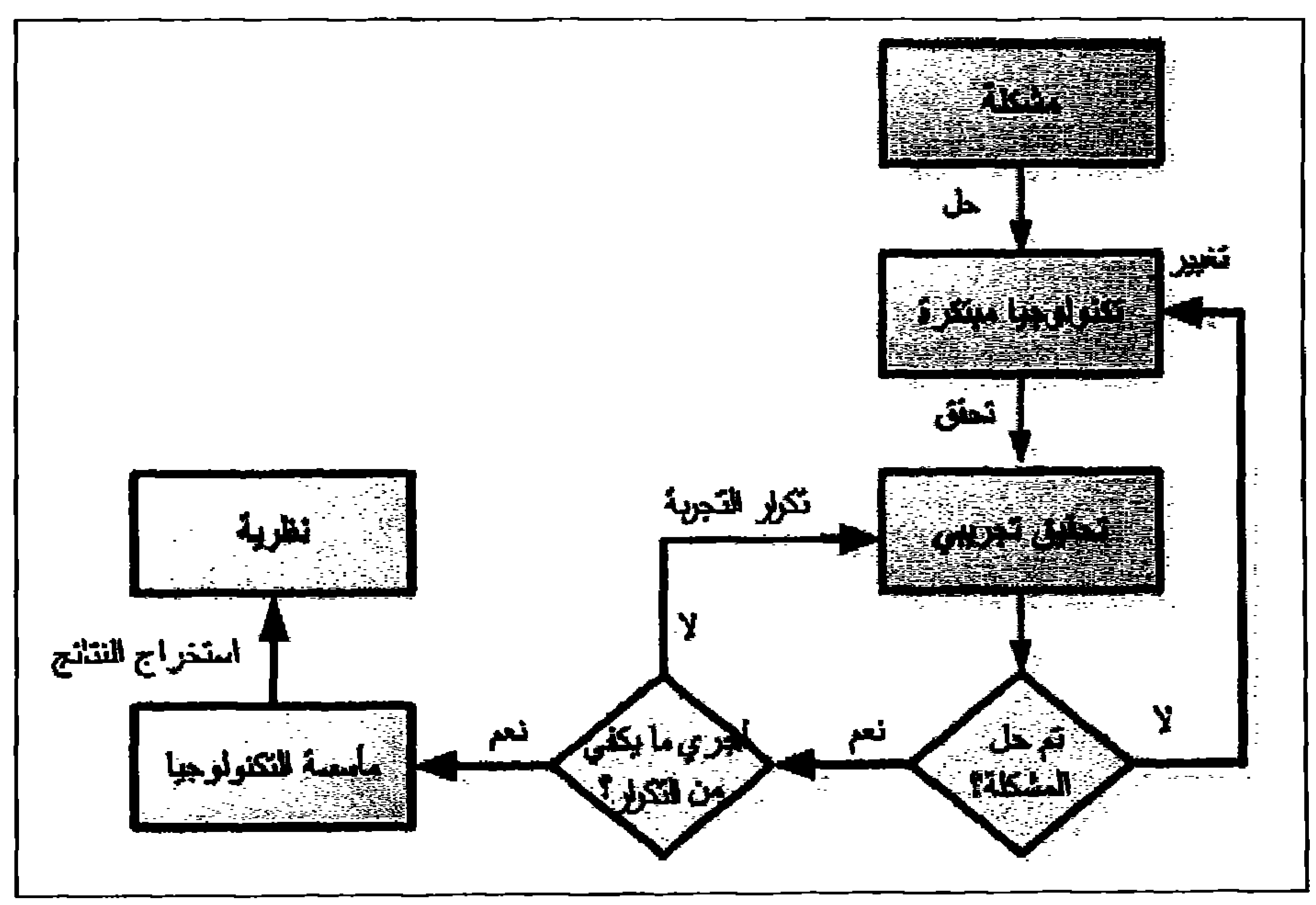
التي تؤدي إلى تأثير سلبي أو تأثير غير مرغوب به وتتبع مسارين بديلين ينفذهما الباحثون. في الحالة الأولى (الشكل 10 ـ 6) يعرّف الباحث نظرية، ويجعلها نافذة المفعول من خلال الأساليب والعمليات والمنهجيات الموضحة كتقنيات مبتكرة ويتحقق منها لإثبات إذا كانت تحل المشكلة. فإذا كانت كذلك، يتم قبول النظرية والتقنيات، وبخلاف ذلك، يتم تعديل النظرية و/أو التقنيات ويستمر الاختبار العلمي. بدلاً من ذلك (الشكل 10 ـ 7)، يحدد الباحث تقنيات مبتكرة جديدة مع الشك في كون التقنيات الجديدة فاعلة في حل المشكلة. يتبع الاختبارات العلمية في حال الحصول على نتائج إيجابية قبول التقنيات وتلخص نتائج الاختبارات في نظرية. وإلا، فإن النظرية يتم تعديلها والتحقق منها مرة أخرى. على سبيل المثال، المعلومات المخفية تكون قائمة على النظرية، لكن الاستقصاء التجريبي ضروري للتحقق من صحة التقنيات التي تجعل منها معلومات تشغيلية في عمليات تطوير البرمجيات؛ ويستخدم النموذج كائني التوجه (object-oriented paradigm) بنجاح كتقنية تسمح لنا بتطوير برمجية ذات جودة عالية مقارنة بنماذج أخرى. مرة أخرى، التحقق من كفاءة نموذج OO تجريبياً غير كافٍ، وكذلك نظرية التلخيص التي لا تزال مفقودة.



الشكل (10 ـ 6): تجميع التكنولوجيا

هندسة البرمجيات وممارساتها هي عمليات محورها العامل البشري. لذا، تعتمد فعاليتها على عوامل التعقيد كالدوافع والخبرة. تعتمد فعالية الممارسات على هذه العوامل وهذه صعبة الفهم كما يصعب التحكم بها. نتيجة لذلك، لا يمكن أن تكون العلاقة بين تطبيقها وتأثير المنتج والعملية قطعية. يمكن تقييم مثل هذه العلاقات تجريبياً[20]. قد تعتمد نتائج الاستقصاء التجريبي على سلوك مواضيع الدراسة التجريبية والمميزات العديدة التي قد تفترضها نفس المشكلة بسياقات مختلفة ومن المتغيرات العديدة التي تكون ضمنية وغير مسيطر عليها.

تشير مظاهر الاستقصاءات التجريبية إلى الحاجة إلى عمل نسخ مماثلة. لذا، تتطلب هندسة البرمجيات أيضاً أن تكون النتائج التجريبية قابلة للنسخ من قبل أطراف خارجية، اتساقاً مع الفكر العلمي الحديث. في المرجع[29]، يشير المؤلفون إلى أن استخدام التجارب الدقيقة التي يمكن إعادتها هو «السمة المميزة للمعرفة العلمية أو الهندسية». يهدف النسخ المتماثل في هندسة البرمجيات إلى ضمان أن العلاقة بين الابتكار والمشكلات التي يحلها والنظرية التي تعزز هذه العلاقة سارية المفعول، هذا من ناحية؛ ومن ناحية أخرى، يهدف إلى تعميم المعرفة التي تأخذ في الاعتبار التغيير الكبير في الظروف التجريبية الصريحة والضمنية.



الشكل (10 ـ 7) : تجميع النظرية

10 ـ 3 ـ 2 تكرار الاستقصاء التجريبي

في عمليات الاستقصاء، التي ينفذها الآخرون، من المهم التأكد من «عدم حصول أخطاء غير مقصودة أو مقصودة»[31]. يمكن أن يساعد التكرار في إقناع المساهمين المحتملين بفاعلية تبنّي ابتكار ما تدعمه نظرية. يمكن الحصول على نتائج تجريبية ذات أهمية مثيرة، على الرغم من أن الحصول على النتائج نفسها في الكثير من الحالات يمكن أن يكون أكثر إقناعاً وبالتالي أكثر قبولاً. إن عمليات الاستقصاء المتكررة التي يتم إجراؤها في حالات مختلفة، التي تؤكد نتائج الدراسة الأصلية والفرضية التي يتم على أساسها التجريب، هي من الطرق التي يتم من خلالها جمع الأدلة. في عمليات التكرار، يجب على الباحث أن يتحكم في المخاطر التي قد تنتج من التجارب في عمليات التحقق. في الحقيقة، يمكن أن تكون نتيجة عمليات التكرار مختلفة عن نتيجة التجربة الأصلية بسبب عدم وجود سيطرة كافية على المخاطر. على سبيل المثال، في تجربة أجراها شنايدمان (Shneidemann) وآخرون حول مدى فائدة الرسم البياني، استنتج أن الرسوم البيانية غير مفيدة في تمثيل الخوارزميات. تم تنفيذ عمليات التكرار لهذه التجربة بواسطة سكانلان (Scanlan)[38]، وقد حدد الخلل في التجربة السابقة على أنه: لم يكن الزمن المخصص لموضوع الدراسة أحد عوامل التجربة الأصلية. كان بإمكانهم أن يستغلوا جميع الوقت الذي يحتاجونه، وأن يستخدموا أي طريقة في عملية التمثيل. أظهر المؤلف أنه يمكن توفير الوقت عن طريق استخدام الرسوم البيانية[38]. تم إثبات ذلك لأن الوقت المسموح كان محدداً. إذاً عن طريق إعطاء الفترة الزمنية نفسها، حصل أفراد التجربة الذين استخدموا الرسوم البيانية لوصف الخوارزميات على نتائج مكتملة بشكل أكبر من الطرق الأخرى. في هذه الحالة، تأثرت التجربة الأصلية بمخاطرة صلاحية البناء.

في عمليات تكرار الاستقصاء، من المهم أن يقوم الباحث بتحديد مقادير التكاليف والمنافع حسب أدوات جمع البيانات. هذه الأدوات مهمة من أجل الحصول على تحليل التكلفة ـ المنفعة للاستقصاء، الذي يمكن أن يكون مفيداً لتكرارات التحقق الأخرى التي سيتم تنفيذها على التجربة. يمكن ملاحظة أنه ينتج من التكرار معلومات مفيدة حول التكلفة والفوائد، وبالتالي زيادة الموثوقية حيث تصبح عنصراً فاعلاً لتكرارات إضافية.

لسوء الحظ، هناك الكثير من الجوانب التي تجعل من الصعب توفير ظروف التجربة الأصلية نفسها في عملية التكرار. وهذه العوامل هي: أولاً، يوجد الكثير من المتغيرات التي تصف سياق الاستقصاء، كما تم ذكره سابقاً. يمكن أن تتغير هذه العوامل مع الوقت إذا تم إعادة التجربة بالسياق نفسه، أو يمكن أن تتغير العوامل أيضاً إذا تغيرت العيّنة موضوع الدراسة نفسها. ثانياً، عادة يتم تنفيذ التكرار لاختبار ما على عيّنات اختبار مختلفة عن العيّنات الأصلية أو السابقة. لذلك، يمكن أن ينتج من التكرار نتيجة تختلف عن النتيجة الأصلية وعن نتائج اختبارات أخرى، نتيجة لاختلاف تخصصات وخبرات العيّنة وليس نتيجة طريقة الإجراء. على سبيل المثال، تخيّل أن المطوّر بخبراته يستطيع أن يحسّن الإنتاجية ست مرات. إذا كانت العيّنات التجريبية التي تكرر عليها الدراسة أكثر خبرة من العيّنة الأصلية للتجربة، يمكن أن تكون النتائج إيجابية، لكن قد تؤثر خبراتهم في النتائج أكثر من تأثير طريقة المعالجة. يضيف هذا الجانب الثاني صعوبة إضافية على تقييد عوامل التكرار: قد تختلف توقعات العيّنات التي تُجرى عليها عمليات الإعادة عن توقعات العيّنة الأصلية التي أجريت عليها الدراسة. يمكن لاختلاف التوقعات هذا أن يؤثر في النتائج.

إن الهدف من الإعادة التي تحصل على نتائج تتوافق مع النتائج الأولية هو تأكيد النتائج الأصلية. إذا كانت نتائج الدراسة الأصلية إيجابية وفقاً لفرضية الدراسة، تعزز الإعادة هذا النجاح، وإذا كانت نتيجة الدراسة الأصلية سلبية، تعزز الإعادة هذا الفشل. وفي المقابل، الإعادة التي تؤدي إلى نتائج تختلف عن نتائج الدراسة الأصلية تحفز لإجراء عمليات تحليل إضافية تهدف إلى تحديد الأخطاء في تصميم وتنفيذ التجربة. من أجل تقييد عوامل عمليات الإعادة كجزء من عملية تسمى محفزات الاختبار[35]، يجب إعداد مجموعة من الوثائق المناسبة التي تصف التجربة الأصلية. تضمن هذه الوثائق وجود دقة كافية في عمليات الإعادة. لذلك، عندما ينتج من عملية إعادة أو أكثر نتائج تدحض نتائج الدراسة الأصلية، يمكن أن يعتمد الباحث أحد الخيارين. إذا تم تأكيد فاعلية الابتكار عن طريق الدلائل، يجب تعزيز النظرية موضع الدراسة قبل قبولها. تعزيز النظرية يعني تعديل أهداف الاختبار أو المسببات أو النتائج أو كليهما.

نتيجة لذلك، تختلف متغيرات وتصميم الاختبار الأصلي، بعد تنفيذ

تغييرات على الاختبار الأصلي، إذا كانت النتائج تتوافق مع توقعات منفّذ الاختبار، يتم إعادة الاختبار لتأكيد النتائج. في الحالة الثانية، إذا لم تثبت فاعلية الابتكار، يجب على الباحث أن يأخذ بالحسبان أنه يجب تعديل الابتكار أو النظرية وفقاً للنتائج التي تم الحصول عليها في التجربة وفي عمليات الإعادة. في الحالتين، يجب تعديل الاختبارات الأصلية وتنفيذها مرة أخرى. عند الحصول على نتائج ناجحة، يجب إعادتها لتأكيد النتائج. في الحالة الأولى تتبع النظرية الابتكار، بينما في الحالة الثانية يتبع الابتكار النظرية.

من أجل إتمام العمل، من المهم تحديد عدد التكرارات المثالية من أجل إعداد مجموعة مناسبة من الأدلة. في حالة التجارب التي يتم التحكم بها يمكن أن نرجع إلى العوامل الإحصائية [25]. في جميع حالات الاختبار الأخرى، من المهم اعتماد خطة موجهة من أجل تحديد العدد المناسب من التكرارات لاختبار ما. يمتلك الاختبار الأصلي أهمية معيّنة، للإعادة الثانية أهمية أكبر لأنها تؤكد نتائج الدراسة الأصلية، للإعادة الثالثة أهمية في زيادة عدد الأدلة على نتائج الدراسة. كلما أصبح عدد الأدلة أكبر، تقل أهمية إجراء تكرارات أخرى. عندما يصبح عدد الأدلة كافياً، تقل أهمية التكرار أكثر فأكثر، بحيث لا تعود ضرورية.

10 ـ 3 ـ 3 الاستقصاءات التجريبية لإنتاج المعرفة

تتطلب إعادة استخدام النتائج التجريبية فعلياً أن يتم تلخيصها وتنظيمها بصورة معرفة يمكن للمهتمين بالنتائج فهمها. يجب أن تكون هذه المعرفة قابلة للاستخدام من دون الرجوع إلى الباحث الذي قام بإعدادها. كذلك، يجب أن تكون قابلة للتطبيق في أحوال مختلفة، وفي الوقت نفسه مصممة للاستخدام في حالة معيّنة[17]. للحصول على هذا الهدف، يجب إعداد مكوّنات المعرفة التي تم تحديدها في الفقرة السابقة. وأكثر من ذلك، يجب أن تحتوي المجموعة المعرفية على جميع المعلومات المهمة لتحديد مدى تشابه أو اختلاف الحالة التي تتم فيها الإعادة عن الحالة الأصلية.

تُعزى الاعتبارات التي تم ذكرها إلى دور الاستقصاءات التي تتم في البيئة الحيوية لإعادة استخدام النتائج التجريبية، التي تعيد التجربة الأصلية بشكل مستمر، وبالتالي إنشاء المعرفة لتحسين وتطوير النظرية التي تم تأكيدها من قبل

الدراسة الأصلية ومجموعة الأدلة عليها. لتحقيق هذا الهدف، قام باسيلي Basili[11, 10] بتوفير مخطط تنظيمي لجمع الخبرات حول إعادة استخدام النتائج التجريبية، من أجل تحليلها وإنشاء مجموعة معرفية حولها. يعرف المخطط باسم معمل الخبرة (EF (Experience Factory، ويعمل على جمع الخبرات وعمليات الاستقصاء التجريبية للمعلومات المرتبطة في عميات التطوير في عدة مجالات: التكلفة والفوائد والمخاطر ومبادرات التطوير. من أجل التوضيح، يصف الشكل 10-8 مخطط معمل الخبرة بشكل مختصر. يمكن الحصول على تفاصيل إضافية في المرجعين[10] و[11] على الرغم من اختلاف المخطط بشكل بسيط. حالما يبدأ المشروع، ويتم تحديد الأهداف، يتم تحديد مصادر وحالات المشروع. تجعلنا هذه المعلومات قادرين على استخراج ملفات الخبرات المتوافرة من قاعدة الخبرات، التي تتضمن المنتجات والدروس المستفادة والنماذج. بوجود أو عدم وجود دعم، يقوم مدير المشروع بتحديد العمليات التي ينوي استخدامها ويستنتج الخطط التنفيذية للمشروع منها. خلال تنفيذ المشروع، يتم تسجيل البيانات التي يجمعها المحلل في قاعدة بيانات المشروع. يتم موالفتها من أجل البحث عن ملفات خبرات أخرى يمكن أن تسهم في تطوير تنفيذ المشروع بما يتناسب مع أهدافه. في نهاية المشروع، يتم مقارنة البيانات التي تم جمعها بملفات الخبرات الحالية ويتم إضافتها وفق الأصول لتوفير مجموعات تحقق إضافية أو لتنقيح محتواها. بعد إنجاز المشروع، يمكن إضافة الدلائل التجريبية الجديدة إلى الموجودة في قاعدة بيانات الخبرات، وبالتالي تعميم أكبر للمعرفة. بهذه الطريقة تصبح قاعدة الخبرات توفّر المعرفة بشكل أعم.

من أجل استخراج المعرفة للنتائج التجريبية، يمكن اعتبار التكرارات التي تشكل ما يسمى عائلات التجارب[12]. تكرر كل تجربة في العائلة تجربة أخرى في العائلة نفسها، مع تعديل في متغيرات الحالة. يتطلب ذلك إطار عمل يوضح النماذج المختلفة والمستخدمة في التحقيقات للعائلة، وقادراً على توثيق القرارات المتخذة خلال التصميم. يجب أن يعطينا الإطار القدرة على تحديد محور الاختبار الذي يجب أن يبقى بلا تغيير هو والعوامل التي تتغير مع كل دراسة في العائلة بحيث يمكن تعميم النتائج في نظرية موحدة توضح جميع أهداف البحث في هذا المشروع.

يشير إطار العمل في هذا الفصل إلى مقاييس السؤال المستهدف (GQM)

التي اقترحها كلٌ من بايسيلي Basili ورومباش Rombach[9] وتم تكييفها لتحدد أهداف المشروع بشكل واضح. في ما يأتي تفاصيل حول إطار العمل الذي تم تكييفه، يليه وصف للتحسينات التي أجراها المؤلف:

يتم تحديد هدف البحث عن طريق خمسة عوامل متغيرة:

1. **هدف الدراسة**: العملية أو المنتج أو أي نموذج تجريبي آخر.

2. **الغرض**: لتمييز/اكتشاف (ما هي؟)، تقييم (هل هي جيدة؟)، التوقع (هل يمكن أن أقيّم شيئاً في المستقبل؟)، التحكم (هل يمكن أن أتلاعب في الأحداث؟)، التطوير (هل يمكنني أن أطور الأحداث؟).

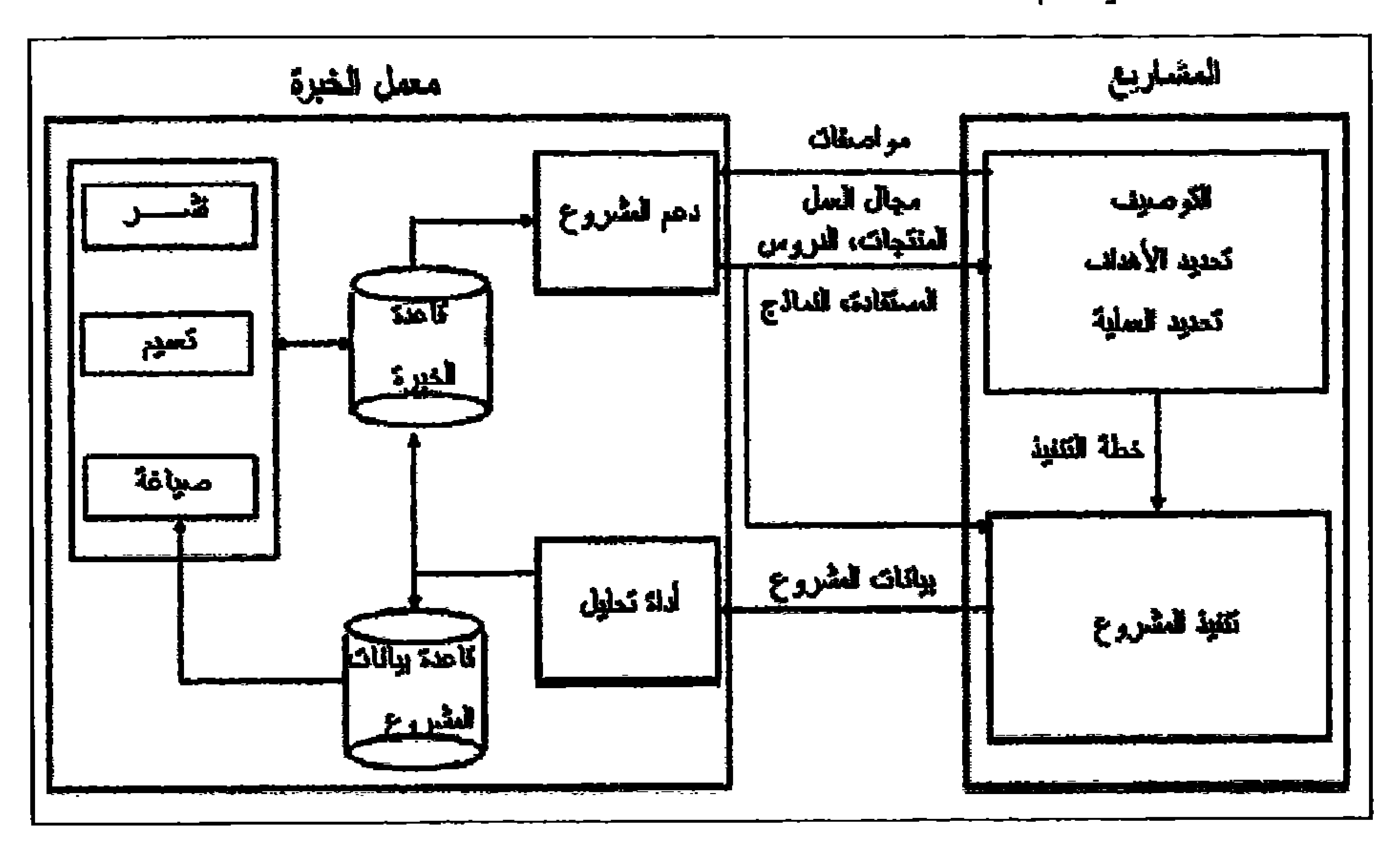
3. **التركيز**: النموذج الذي يهدف إلى عرض نواحي هدف الدراسة المعنية ـ على سبيل المثال، التأثير في مدى موثوقية المنتج، تحديد الخلل/منعه، وقدرات العملية/دقة نموذج التكاليف.

4. **وجهة النظر**: ما هي نواحي التأثير التي نريد أن نحصل عليها؟ على سبيل المثال، وجهة نظر الباحث الذي يحاول الحصول على بعض المعلومات حول النقطة المهمة.

5. **السياق**: النماذج التي تهدف إلى وصف البيئة التي تم فيها أخذ القياسات (على سبيل المثال، الطلاب/المشاركون، المبتدئون/الخبراء، في البيئة الحيوية/في المختبر؛ المشاريع يجب تنفيذها في أثناء الاختبار/مشاريع تم تنفيذها بالفعل).

لتوضيح عوامل أهداف البحث بصورة أفضل، يجب التصريح عن القيم التي تم افتراضها في الاختبار. يمكن أن يكون هدف الدراسة عملية تتطلب كفاءات تقنية لإجراء مهمات معيّنة (تقنيات)، أو عملية تصف كيفية إدارة تقنيات التطبيق للحصول على هدف ما (أسلوب)، أو عملية تصف التطور الكلي لبرنامج ما (دورة الحياة). إن الهدف العام للاختبار هو تقييم شيء ما. يريد الباحث دراسة تقنية ما وتقييم تأثيرها في مرحلة التطوير. أكثر وجهات النظر تكراراً هي وجهة نظر الباحث أو معد ملف المعلومات. عادة ما يتضمن اختبار المشروع وجهات نظر المدراء والمهتمين بتقييم أثر التقنية في الجهد المبذول والمواعيد الزمنية والعائد من الاستثمار. عندما يكون هدف الدراسة تقنية ما، غالباً ما يكون التركيز منصباً على تقييم أثر التقنية في مكوّنات

برمجية واحدة أو أكثر، سواء كانت عملية أو منتج أو مبرمجون. يتضمن سياق البحث عوامل بيئية متعددة. يمكن إجراء الدراسات على طلاب أو خبراء، بوجود أو بعدم وجود قيود على الفترة الزمنية، في مجالات التطبيق المعروفة أو التي لم تُدرس بعد.



الشكل (10 ــ 8): معمل الخبرة

يمكن أن يتضمن هدف استقصاء ما على عدة عوامل؛ في هذه الحالة، يجب تصميم الدراسة الجيدة لكل مجموعة من القيم التي يمكن أن تنتجها العوامل. إضافة إلى ذلك، يتم ربط كل دراسة بهدف محدد. وفقاً لذلك، تكون مهمة الباحث جمع نتائج كل دراسة وتعميمها بحيث يمكن استخراج المعلومات من عائلة التجارب.

هذا مثال على عائلة ما: يتضمن هدف الدراسة أسلوبين لصيانة النماذج (إصلاح سريع (Quick Fix) (QF)، وتحسين تكراري (Iterative Enhancement) (IE))؛ الهدف هو تقييم الفرق بين الطريقتين QF و IEمع التركيز على تقييم تأثير هذه الطرق في مجموعة من الخصائص fQ1, Q2, Q3, Q4g، من وجهة نظر مدى فاعلية تأثير العملية في مجموعة الحالات fCxg. إن Cx هو المتغير الذي يجعل من الهدف السابق عائلة من العوامل. لهذه النقطة، يجب تفصيل القيم التي يتم افتراضها.

س1: التصحيح

مجموعة المكوّنات المعدلة: MCS

مجموعة المكوّنات المتوقعة: ECS

مجموعة المكوّنات الصحيحة: CCS

التصحيح: COR = #(CCS

س2: الإتمام

الإتمام: COM = #(MCS

س3: تعديل الخطة الزمنية

الزمن اللازم لتصميم التعديل: Tdes

الزمن اللازم لبرمجة التعديل المصمم: Tcod

الزمن اللازم لإجراء عمليات الإصلاح: T = Tdes + Tcod

س 4: القابلية على التتبع

أفراد العيّنة التجريبية التي يمكن تتبعها بشكل محمي: Ns

أفراد العيّنة التجريبية: N

القابلية على التتبع: TR = Ns/N * 100

C1: طلاب الجامعات (Un.St.) الذين يتعاملون مع تأثير التعديل ذي المستوى المنخفض.

C2: طلاب الجامعات (Un.St.) الذين يتعاملون مع تأثير التعديل ذي المستوى المرتفع.

C3: المحترفون (Prof) الذين يتعاملون مع تأثير التعديل ذي المستوى المنخفض.

C4: المحترفون (Prof) الذين يتعاملون مع تأثير التعديل ذي المستوى المرتفع.

لأسباب تتعلق بالمساحة، لن يتم توفير وصف عملية التجربة وتنفيذها. يبيّن الجدول 10-14 ملخصاً للنتائج. من أجل التوضيح، بإعطاء نموذج معيّن مثل Pi، تظهر الإشارة «+» قبل النموذج إذا كانت قيمته أكبر من قيمة المنافس له، أما إشارة «++» فتشير إلى زيادة ملحوظة في قيمته بالنسبة إلى النموذج المنافس له في الدراسة قيد البحث.

تلخيصاً للنتيجة، يعبَّر عن كفاءة نموذج الصيانة بالرمز T، ويتم التعبير عن الفاعلية عن طريق مجموعة أخرى من عوامل الجودة التي يتم قياسها. لذلك، المعرفة التي يمكن تلخيصها عن طريق تحليل النتائج المبيّنة في الجدول 10-14 هي: IE أكثر كفاءة من QF، IE بشكل عام أكثر فاعلية من QF عند مقارنة التعديلات الطفيفة. لكن، ثمة خطورة عالية في عمليات التتبع، لذلك من الأرجح أن ينتج منها انخفاض في نوعية التطبيق البرمجي. هذه المعرفة التجريبية تدحض الدلائل السابقة والتي هي، وفقاً للمطوّرين، QF أكثر كفاءة من IE.

تشير النتائج أيضاً إلى أن الاستدلال يهمل الكفاءة المنخفضة ل QF في ما يتعلق بضمان جودة النظام البرمجي. إضافة إلى ذلك، تشير الخبرة إلى أن نموذج الصيانة يكون أكثر كفاءة عند تطبيق IE على المحترفين عنها عند تطبيقها على الطلاب. أخيراً، تشير الدلائل أن QF تكون أكثر كفاءة فقط عندما يكون تأثير التعديل طفيفاً.

الجدول (10 ـ 14)

ملخص نتائج التجربة

السيـــاق							
TR	T	COM	COR	تأثير التعديلات	الخبرات	عدد التكرارات	مكان التجربة
+IE	+QF	+IE	+IE	منخفض	طالب جامعي	3	جامعة باري
+ +IE	+IE	+IE	+IE	مرتفع	طالب جامعي	3	جامعة باري
+IE	+QF	+ +IE	+IE	منخفض	محترف	1	P.A
+ +IE	+IE	+ +IE	+IE	مرتفع	محترف	1	P.A

10 ـ 4 الاستقصاء التجريبي لقبول الابتكار

كون الإنسان هو محور هندسة البرمجيات، يمكن عرض الابتكار ونشره من خلال عمليات حقيقية إذا تم قبول هذا الابتكار من قبل المطوّرين المسؤولين عن العمليات. يمكن توقع النظرية الآتية: المعرفة التي يمكن أن يوفرها الابتكار ويفيد بها عملية تطوير البرمجيات هي شيء ضروري، على الرغم من أنها ليست كافية للابتكار الذي سيقبل به المطوّرون، لأنها تعتمد على عوامل اجتماعية واقتصادية لعملية التجريب. القبول هو العامل الأساسي لتعميم الابتكار وبالتالي وجود طلب عليه. لأخذ النظرية السابقة بالاعتبار، من المفيد مشاركة المطوّرين الذين سيقومون بشكل مؤكد بالتعامل مع الابتكار كعيّنة تجريبية. النتيجة المتوقعة هي أن يستوعب المشاركون المعرفة الجديدة، وبالتالي يقوموا بتعميمها وجعلها صالحة للنشر.

استخدم المؤلف خبرته في تنفيذ EI في الشركات لتأكيد النظرية السابقة. في هذا الفصل، ولغايات تتعلق بالمساحة، لم يتم سرد التفاصيل. ولكن، يمكن أن يطالع القراء المهتمين المرجعين[4] و[41] كمراجع للخبرات المطلوبة، التي قد تكون مفيدة للباحثين الآخرين حول اختيار طرق الاختبار المناسبة أو حول تأكيد النظرية السابقة.

تمّ تصنيف كل دراسة تم تحليلها حسب واحدة من الفئات التي تم سردها وتصنيفها وفقاً لعوامل تنظيمية وتقنية، تم وصف ذلك في جدول 10-15، في تدريج مرتب (H = مرتفع، L = منخفض). يظهر الجدول 10-16 نتائج عمليات التجريب. هذا ويمكن اعتبار أمور أخرى.

في الاستقصاء بأثر رجعي، تم إشراك المطوّرين الذين شاركوا في البرامج المنفذة كأفراد في العيّنة التجريبية من دون إدراكهم. بهذه الطريقة فقط يمكنهم تعلم ومشاركة الأمور السليمة المستخدمة مع تفسيراتها؛ أي إنهم يدركون فوائد الابتكار. لكن نظراً إلى أنه لا يتم إشراكهم في استخدام المشروع، لذا فإن أحد الشروط الأساسية لعملية القبول غير متوافر. استراتيجية الاختبار هذه، ووفقاً للنظرية التي تم ذكرها سابقاً، لا تفرض شروطاً لقبول الابتكار. تؤكد الخبرة التجريبية هذا النقص: جميع المطوّرين بمن فيهم أولئك الذين قاموا بتنفيذ المشروع، لا يستوعبون الابتكار.

الجدول (10 ـ 15)

عوامل التوصيف

متغيرات العامل التنظيمي	الاستثمار	ما تستثمره الشركة لتقديم الابتكار	مرتفع (H)، عندما يحدث نقل التكنولوجيا خلال أكثر من 6 أشهر. من الصعب التوقع بتأثير الابتكار في عمليات الإنتاج في الشركة.
	العائد على الاستثمار	العائد على الاستثمار الذي تحصله الشركة بعد تقديم الابتكار.	مرتفع (H) عندما يكون مقداره أكبر بثلاث مرات من الاستثمار في غضون عامين على الأقل.
	MANAG Imp	التحسينات المتصورة من قبل الإدارة.	تقييم ذاتي معبر عنه من خلال اعتبارات شخصية وإجابات يتم تحصيلها من خلال المقابلات أو الاستبيانات.
متغيرات العامل التقني	Dev_Imp	التحسين المتصور من الفريق التقني باستخدام التقنية الجديدة.	تقييم ذاتي معبر عنه باعتبارات شخصية وإجابات يتم الحصول عليها من خلال المقابلات والاستبيانات.
	Diff_QM_Def	الصعوبة التي يواجهها المبتكرون لتحديد نموذج الجودة المستخدم لتقييم فعالية وكفاءة الابتكار.	مرتفع (H) عند إصدار ثلاثة إصدارات كبرى من نموذج الجودة على الأقل في أثناء الاستقصاء قبل الحصول على الإصدار الأخير.
	Diff_QM_Acc	الصعوبة التي يواجهها المساهمون في قبول نموذج الجودة المستخدم لتقييم فعالية وكفاءة الابتكار.	مرتفع (H) عند عقد ثلاثة اجتماعات كبرى يدعو لها المساهمون للحصول على مزيد من التوضيح عن نموذج الجودة وتطبيقاته.

يتم تقييم استقصاء المشروع وفقاً لنوع الاستقصاء. في حالة الاستقصاء، يتعلم المطوّرون استخدام الابتكار لكن نادراً ما يكونون مقتنعين بفوائده، على الرغم من أنه يكون موافقاً على التدابير المتخذة والتفسيرات التي تم الحصول عليها. وفقاً للمؤلف، يعود سبب ذلك إلى أن المطوّر يرى تصميم التجربة كما لو أنه فرض عليه، ولا يشعر أنه مشارك فيه. للاستقصاءات الميدانية عدة متغيرات: عادة ما يتم تقييم الاستقصاء التجريبي خلال عملية الاستقصاء وبمشاركة المطوّرين الذين يستوعبون النتائج والتقنيات المستخدمة. يتم الحصول على النتائج نفسها من الاستقصاءات التمهيدية، مع وجود فرق في أن الموضوع قيد التجريب يتم مراقبته ولا يتم مشاركته. لذلك يبدو الابتكار قيد التجريب غريباً بالنسبة إلى عيّنات الاستقصاء. لا تؤدي الاستقصاءات التمهيدية في الميدان إلى التأثيرات نفسها لأن العيّنة يتم مراقبتها ولا يتم مشاركتها.

الجدول (10 ـ 16)

ترتيب المعاملات في الاستقصاءات المختلفة

المميزات	استقصاء بأثر رجعي		دراسة حالة						تجربة	
			دراسة حالة تجريبية		دراسة حالة تمهيدية		دراسة ميدانية		تجربة محكومة	
الوحدات التجريبية	البنك الإيطالي ـ س	البنك الإيطالي ـ ص	البنك الإيطالي ـ ص	البنك الفرنسي	الإدارة العامة	Italian SME-Z	Italian SME-W	Multina-T ENT.	Univ & SME	Univ
استثمار	L	L	H	H	H	H	H	H	L	L
العائد على الاستثمار	L	L	H	H	L	H	H	H	L	-
MANAG I,mp	H	H	H	H	H	H	H	H	L	L
Dev Imp	-	-	L	L	L	H	H	H	L	L
Diff_QM_Def	L	L	H	H	H	L	L	L	H	H
Dif_QM_Acc	H	H	H	H	H	H	H	H	L	L

ملاحظة: H = مرتفع، L = منخفض

في تجربة ما، إذا تم تحفيز المطوّرين بشكل جيد وأعجبوا بالنتائج، سوف يتطلعون إلى فوائد الابتكار. مع الأخذ بالاعتبار قصر فترة تنفيذ الاختبار. لا يستوعب المطوّرون الابتكار بشكل كافٍ. وفقاً للفرضية السابقة، لا يستطيع هذا النوع من الاختبار تعميم الابتكار.

10 ــ 5 بناء الكفاءات من خلال الاستقصاء التجريبي

10 ــ 5 ــ 1 مقدمة عامة

تكمن أهمية EI في أنه تدريب يقوم بتعزيز القيمة المعرفية لهندسة البرمجيات وجعله مقبولاً بشكل واسع. تحديداً، تحفز EI النمو المعرفي وتطوير القدرات المطلوبة في الدرايات العملية، من أجل تأكيد النماذج المعروفة. إضافة إلى ذلك، تشير EI إلى مواقع الضعف في المعرفة السابقة، وتحفز لإنشاء معرفة جديدة، وتركز على المشاكل التي ظهرت في عملية تحويل الابتكار من الحالة النظرية إلى الحالة العملية. كذلك، تقيم المنهج المتبع في كل عملية نقل. يلخص المؤلف في الفقرات الآتية الكفاءات التي تم الحصول عليها في عمليات الصيانة الاستثنائية خلال 10 أعوام من الاستقصاءات التجريبية لمشروع SERLAB. يمكن الحصول على معلومات إضافية من المرجع[6].

لأجل التوضيح، سيتم تعريف بعض المصطلحات التي سيتم استخدامها. تعرّف الكفاءة على أنها القدرات والمعرفة التي تجعل من الممكن إدارة الموارد المتاحة من أجل الوصول إلى الأهداف التي تم تحديدها بشكل منهجي. وفقاً للمرجع[36]، يتم الحصول على المعرفة من خلال فهم المعلومات والعلاقات بينها وتصنيفها الصحيح. يتم تعريف القدرات على أنها تطبيق للمعرفة بشكل عملي تهدف إلى إيجاد الحلول لمهمة ما في بيئة حقيقية.

يتناول مسح الاستقصاء التجريبي الحالي الصيانة الاستثنائية (EM). تشير الصيانة الاعتيادية إلى التعديلات التي تهدف إلى التغلب على السلوك الذي لا يتوافق مع المتطلبات، أو تهدف إلى جعل النظام كافياً ومناسباً للتطبيق وللابتكارات التقنية. أثبت ليمان Lehman[21, 22] بشكل عملي أن الصيانة العادية تقلل من جودة النظام، وتتطلب عملية الصيانة عندئذ زمناً أكبر، وتصبح البرمجيات أقدم. تشير الصيانة الاستثنائية إلى التعديلات التي تهدف إلى إبطاء تقادم البرمجيات، وبالتالي الحفاظ على قيمتها الاقتصادية.

توفّر المادة النظرية عمليات أكثر من العمليات التي تمّت مناقشتها في التحليل التالي. هذا يتطلب كفاءات أكبر، على الرغم من أنها جزء من خبراتنا. يتم توضيح تفاصيل أخرى هنا:

● الهندسة العكسية (Rverse Engineering) (RE)، التي تمكننا من الحصول على وصف للنظام عند مستوى تجريدي مرتفع بدءاً من الوصف المنخفض المستوى الذي يتم تزويده[15].

● إعادة التصميم، الذي يمكننا من الحصول على شكل جديد لبعض المنتجات، مع وجود جودة عالية ومستمرة في عملية تطبيق المنتجات[15].

● يمكن إضافة عملية الاستعادة إلى القائمة السابقة وفقاً لخبراتنا، وهي تمثل البديل لعملية إعادة البناء التي تتضمن إعادة بناء الشيفرة البرمجية الأصلية من الحصول على نسخة جديدة تحقق مبدأ البرمجة الهيكلية اعتماداً على الخصائص الهيكلية للشيفرة البرمجية. يتضمن البديل أيضاً مراعاة عملية إعادة الهيكلة للشيفرة البرمجية[41, 42].

تمت الإشارة بشكل أوسع إلى قدرات العملية السابقة في المرجع[38]، في حين تدفعنا الخبرات التي تم تطويرها في المشاريع الصناعية عادة إلى الإجابة عن الأسئلة الآتية:

س1: متى يجب استخدام كل عملية؟

س2: لماذا يجب استخدام عملية محددة؟

لتحسين مستوى قبول عمليات الصيانة الاستثنائية، يمكن إضافة الأسئلة الآتية:

س3: ما هي التكاليف والمنافع الناجمة عن استخدام العملية؟

س4: ما هي المخاطر التي تتضمنها، وكيف يمكن التقليل منها؟

تم إعداد حزمة معرفية استخرجت من تحليل البيانات وتعميم النتائج التي تم جمعها خلال عملية الصيانة الاستثنائية المطبقة على النظام، لتوضيح ذلك بصورة أكبر سوف نرمز للبنك الإيطالي الذي تطبق عليه هذه العملية[42] بالرمز X (الجدول 10-16). جميع المعلومات التجريبية الموجودة في هذا القسم والقسم الذي يليه تتعلق بنظام معلومات البنك، إلا إذا تم توضيح غير ذلك. هناك تفصيل أكبر في المرجع[14].

يتضمن نوع الصيانة الاستثنائية التي أنتجت المعلومات في هذا القسم وفي القسم الذي يليه اختبارات سابقة تم تنفيذها على المعلومات التي تم جمعها خلال عملية الصيانة الاستثنائية لنظام المعلومات السابق. تتطلب نتيجة الاختبار عمليات تحقق أكثر من خلال إعادة التجارب مماثلة. لذلك يتم الإشارة إليها على أنها دروس مستفادة.

10 ـ 5 ـ 2 أعراض التقادم

للبدء في الظروف التي تتطلب إجراء عمليات صيانة استثنائية، من المهم تحديد خصائص البرمجيات التي تنكشف عند الضرورة. وفقاً لتجربة SERLAB في عملية الصيانة الاستثنائية أن يتم التعبير عن هذه الخصائص من خلال مؤشرات، وتحدد هذه المؤشرات على أنها مؤشرات تقادم. تالياً، ثمة قائمة غير شاملة يمكن توسعتها وفقاً لعمليات الاستقصاء التجريبية الأخرى.

التلوث (Pollution): العديد من مكوّنات النظام، بما فيها البيانات والوظائف، مشمولة في البرمجية، على الرغم من كونها غير مفيدة لاحتياجات النظام. هذه الأعراض تجعل من نشاطات الصيانة أكثر صعوبة وأكثر تكلفة نتيجة عدم التحديد الفاعل للمكوّنات التي تتأثر بالتعديل. نتيجة لذلك، تكون الصيانة أقل موثوقية. المعايير المستخدمة في هذه المؤشرات هي:

- البرامج المكررة: البرامج التي تظهر عدة مرات بمعرفات مختلفة في مكتبات البرنامج للنظام البرمجي.
- البرامج غير المستخدمة: البرامج في مكتبات النظام البرمجي التي لا تستخدم نتائجها من قبل مستخدمي النظام.
- البرنامج في المكتبات: البرامج التي تنتمي إلى مكتبة المكوّنات للنظام.

المعرفة المضمنة (Embedded Knowledge): يتم تضمين المعرفة لمجال التطبيق في البرمجيات، كتأثير للصيانة السابقة. يتم دمجها بالوثائق التي لا يمكن تتبعها باستخدام البرمجية. لا يمكن إعادة استخدام المعرفة المضمنة من قبل منفذي عمليات الصيانة، وبالتالي تصبح عمليات الصيانة غير موثوقة بشكل أكبر لأنه لا يمكن تحديد تأثير التغييرات بشكل مؤكد. هناك العديد من المعايير التي يمكن من خلالها معرفة مقدار هذه المؤشرات وتعتمد على المستوى المعرفي للنظام. في حالتنا هذه تم استخدام الآتي:

■ عدد الوظائف التجارية المستخدمة: عدد الخدمات أو الوظائف التجارية التي يستطيع المستخدمون طلب النظام من أجلها.

■ عدد الوظائف التي يمكن تتبعها في البرمجية: عدد الخدمات والوظائف التجارية، المعروفة من قبل المستخدمين، التي يمكن تتبعها من خلال وثائق النظام أو الشيفرة.

■ قاموس ضعيف: أسماء البيانات غير مترابطة مع المعنى الذي تحمله للمتغيرات والبرمجيات المرتبطة بها، مما يؤدي إلى صعوبة في فهمها. هذا يتطلب جهداً أكبر في عمليات الصيانة.

التقارن (Coupling): يوجد الكثير من العلاقات التي تتخلل مكوّنات النظام أو البرمجية، وبالتالي تصبح البيانات الناتجة والقدرة على التحكم أكثر صعوبة للفهم والإدارة. يزود الإطار النظري عدة معايير لعمليات التقارن. تالياً، تم استخدام عدد محدد من عمليات التقارن الداخلي لبرنامج ما بين إجراءات بيانات نظام معقد مثل التي تم استخدامها في التجربة. لتقييم التقارن بين البرمجيات، تم استخدام المعايير الآتية:

■ برنامج مالك الملف: يعرف البرنامج Pi الذي يتبع النظام البرمجي S على أنه مالك البرنامج Fi إذا كان يحتوي الوظائف اللازمة لتعديل محتويات Fi.

■ الملفات الباثولوجيكية: يسمى Fi ملفاً باثولوجيكياً (Pathological) إذا كان هناك برنامجان أو أكثر في النظام S قادرة على إجراء تغييرات على Fi.

إذا كان هناك برنامجان يمتلكان الملف نفسه، يتم تقارنهما بشكل صارم لأن كليهما يعرف بنية الملف ومحتوياته. يؤثر كل تعديل في الهيكلية أو في المحتويات في جميع البرامج التي تمتلكها، وقد يختلف التأثير من برنامج إلى آخر.

عندما يحتوي النظام على تقارن مثل هذه يكون له تأثير في كل تعديل يتم إجراؤه على الملفات الباثولوجيكية. بالإضافة إلى ذلك، بما إن محتويات الملف الباثولوجيكي مرتبطة بسلوك عدة برامج، يجب وضع اختبار يكون قادراً على فحص سلوك برنامج واحد أو عدة برامج فقط. في الحقيقة، يؤثر كل سجل للملف الباثولوجيكي في البرامج الباثولوجيكية بشكل مختلف.

الهيكلية الضعيفة (Poor Architecture): في العادة يتم تنفيذ عمليات الصيانة التي تتم خلال فترة عمل النظام وفقاً لمناهج بنائية مختلفة، وفي بعض

الأحيان تتعارض مع بعضها البعض. يؤدي ذلك إلى اتخاذ قرارات غير مناسبة خلال عمليات الصيانة تؤدي إلى إضعاف هيكلية النظام، وبالتالي زيادة الجهد اللازم للصيانة. المعايير المستخدمة للإشارة إلى هذه العواقب التي تنتج من القرارات وتحد من مقدار المؤشر هي:

- ملفات غير مستخدمة: الملفات التي تم إنشاؤها وتحديثها وإلغاؤها من دون قراءتها.
- الملفات المتقادمة: الملفات التي لم يسبق تحديثها وحذفها.
- الملفات المؤقتة: الملفات التي تم انشاؤها وقراءتها، ولكن لم يسبق تحديثها وحذفها.
- البيانات ذات الدلالات المكررة: البيانات التي تكون المجالات المحددة لها مماثلة المجالات المحددة لبيانات أخرى أو تكون محتواة فيها.
- البيانات المكررة حسابياً: البيانات التي يمكن حسابها بدءاً من بيانات أخرى في قاعدة البيانات نفسها.

10 ـ 5 ـ 3 الهندسة العكسية

عندما تبدأ عملية الصيانة الاستثنائية، كان العمر المتوسط للبرنامج 12 عاماً تقويمياً، لكن أساسها قد كتب قبل ذلك بـ 23 عاماً. لمنع إهدار الجهد أو عمليات المعالجة التي تم تطبيقها على النظام كان الفحص بهدف تحديد والقضاء على أسباب التلوث. النتائج موضحة في الجدول 10-17.

الجدول (10 ـ 17)

قيم مقياس التلوث التي تم قياسها قبل وبعد المعالجة في الهندسة العكسية

مقدار التلوث	قبل الفحص	بعد الفحص
برامج مكررة	4.323	0
برامج غير مستخدمة	294	0
برامج في المكتبات	6.508	1.891

من الواضح كيف أن القضاء على التلوث، على الرغم من كونه يدوياً، قد قلّل من موجودات البرمجيات الفعلية التي يجب التعامل معها عن طريق الصيانة الاستثنائية، فهي توفّر في إجراءات عملية الصيانة الاستثنائية. يمكن أن ينتج

التلوث نتيجة عمليات الصيانة الاعتيادية. لذلك، أول الدروس المستفادة هي:

فحص البرمجية بهدف تجريدها من موجوداتها السابقة والناتجة من تطبيق المشاريع يعترض ويقضي على كم كبير من التلوث.

إذا تم إجراء هذا الفحص على أسس اعتيادية، فإنها تحتاج إلى جهد أقل، ويمكن استخدامها لمراقبة النظام البرمجي وتفادي وجود مكوّنات غير مفيدة.

تم تنفيذ أولى الخطوات لتحديد المعرفة المضمنة من خلال مسح طُبّق على منفذي عمليات الصيانة وعلى المستخدمين كعيّنات اختبار. خلال المقابلات طلب منهم وصف وظائف الأعمال التي كانوا يعلمون عنها في النظام البرمجي الذي استخدموه أو قاموا بصيانته:

- عدد وظائف الأعمال المستخدمة: 935.

تم اعتماد عملية الفحص لتحليل عدد الوظائف القابلة للتفسير في الوثائق والشيفرة البرمجية للنظام البرمجي.

- عدد الوظائف التي يمكن تتبعها في البرمجية: 25.

كان من غير الممكن تحديد البرامج التي توفر وظائف النظام الـ 910 الأخرى.

الدرس الذي تم تعلمه هو:

- الحصول على المعرفة من البرامج يكون مكلفاً وذا موثوقية منخفضة.

من الضروري تحديث الوثائق للحفاظ على قابلية تتبع البرنامج. لذلك، من الضروري إجراء فحص مستمر للقابلية على التتبع.

يظهر الاستدلال المتفق عليه بشكل عام بأن عملية الهندسة العكسية تهدف إلى وصف النظام عند مستوى مرتفع من التجريد، حتى يتم فهمه بشكل أكبر. لذلك، تم تنفيذ الهندسة العكسية.

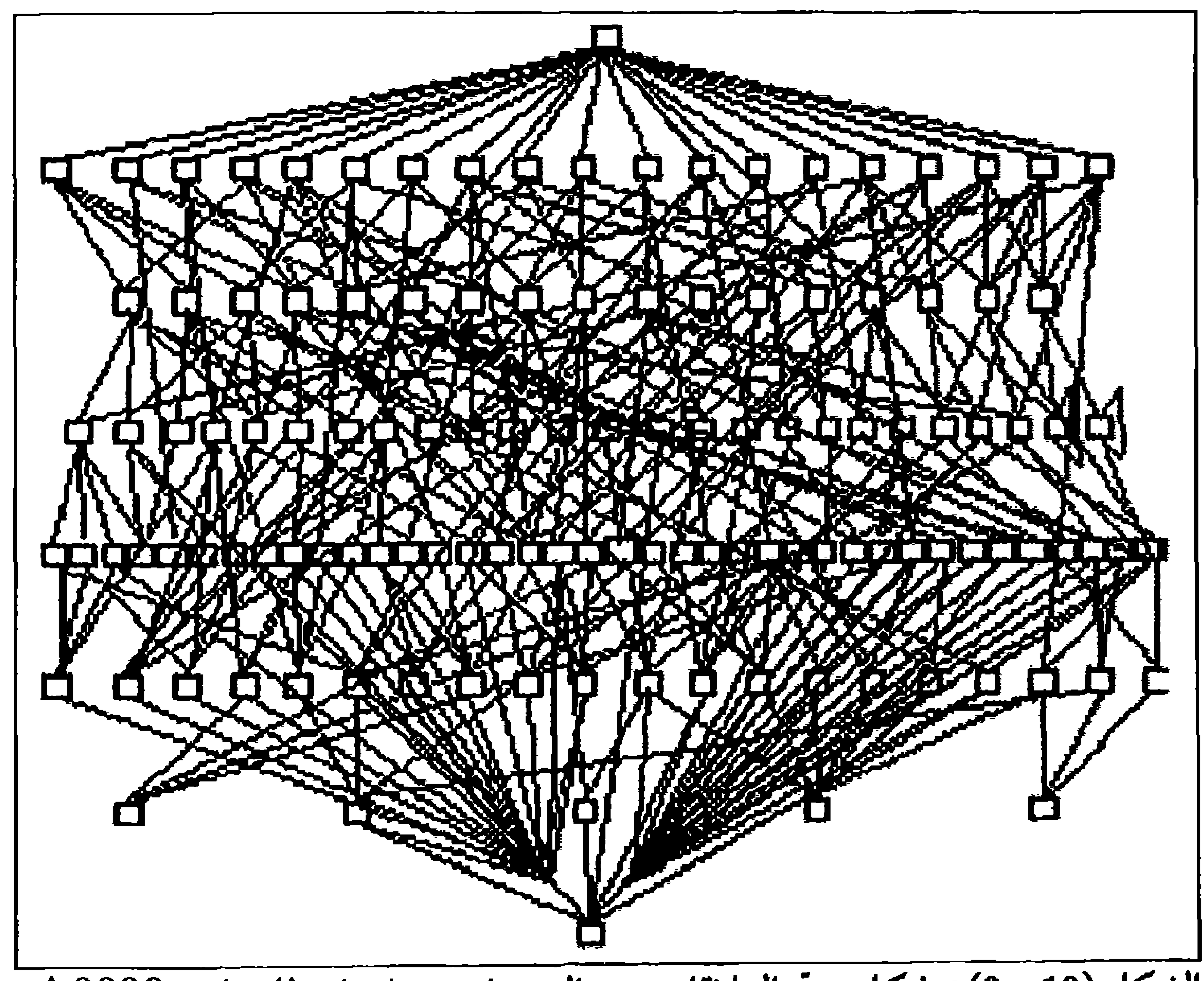
استخدمت العملية اثنتين من وظائف الأعمال لتنفيذ مهمتين حرجتين: الأولى من أجل إجراء الهندسة العكسية للبرامج، والثانية من أجل إجراء الهندسة العكسية لقاعدة البيانات. عادة ما تُعتمد الوظائف التي تم اختيارها في بيئات مختلفة وحديثة وذات سمعة جيدة في المجتمعيّن التجاري والأكاديمي. إضافة إلى ذلك، كانتا ذواتي تكلفة عالية عند شرائهما وعند استخدامهما. لسوء الحظ أصبحتا غير ملائمتين: تم إيقاف الوظيفة الأولى عندما تخطى عدد نقاط القرار في البرنامج الحد المعطى، هذا الحد كان أقل من عدد نقاط القرار في

التطبيق. الأداة الثانية كان لها سرعة استجابة كبيرة (بحدود 5 أيام!) عندما تخطت المعلومات المراد تحليلها الحد المعطى. كان الحد المعطى أقل من المعلومات المراد تحليلها أيضاً في هذه الحالة.

لم تصل معالجة عملية الهندسة العكسية إلى هدفها وهو تأكيد فهم البرمجية. في الحقيقة، لم تساعد الوثائق التي تم الحصول عليها في فهم البيانات والبرنامج، والسبب الأساسي لذلك هو أن الجودة التقنية انخفضت بشكل كبير. التفاصيل وعمليات الفحص تتوافر في المرجع[42]، هذا ملخص لبعض النواحي المهمة:

■ 75 في المئة من البرامج لها قيم محددة ما بين 75 ـ 185 (تقريباً)، أي أنها أعلى من الحد الموصى به من قبل مهندسي البرمجيات.

■ تضمّن العديد من البرامج وحدات تمّ تقارنها بشكل صارم. يبيّن الشكل 10-9، كمثال، العلاقة البيانية بين وحدات البرنامج (الأكثر أهمية في النظام البرمجي، لأنها تهدف إلى تتبع إجراءات المستخدمين وتتطلب خدمات مناسبة وكافية للنظام).



الشكل (10 ـ 9): شكل يبيّن العلاقات بين الوحدات تم انشاؤه للبرنامج A0000

الدروس المستفادة هي:

- عملية الهندسة العكسية فاعلة في الحصول على وصف ذي مستوى مرتفع، بدءاً من مستوى تجريدي منخفض، لكن مستوى جودة الوثائق يرتبط بشكل كبير بمستوى جودة المكوّنات التي تمّت معالجتها. لذا فإن عملية الهندسة العكسية تكون غير قادرة على تحسين عملية الفهم، بدلاً من ذلك، يكون الهدف من عملية الهندسة العكسية هو الحفاظ على قابلية التتبع بين مستويات تجريدية مختلفة للنظام، بعد تنفيذ عمليات الصيانة في مرحلة التشفير. على سبيل المثال، عندما تقوم عملية الصيانة بتغيير الشيفرة البرمجية لتصبح هيكليتها جيدة ولا ينتج منها أي ضعف كبير، يمكن بعد ذلك استخدام عملية الهندسة العكسية لإعادة بناء هيكلية البرنامج بعد انجاز عملية الصيانة. من المهم مراقبة مستوى فهم الوثائق، نتيجة لضعف في الشيفرة البرمجية، وبالتالي مستوى فهم الوثائق الذي ينتج من عمليات الصيانة[21, 22].

- تتضمن عملية الهندسة العكسية عدة نشاطات، التي يمكن وصفها بشكل رسمي، وبالتالي دعمها عن طريق الأدوات الأوتوماتيكية، ما يؤدي إلى انخفاض تكاليف العمليات.

- يوجد خطورة في عملية اختيار الأدوات لأنها قد تكون غير مناسبة للنظام، ويجب أيضاً وضع هذه الخطورة بالاعتبار حتى عندما يكون لهذه الأدوات سمعة جيدة في السوق، ولم يتم اختبار فعاليتها من خلال اختبار الإجهاد على عيّنة مناسبة من البرامج الحقيقية.

10 ـ 5 ـ 4 الاستعادة

من الحلول الممكنة للتغلب على المعرفة المضمنة إعادة هيكلة النظام السابق قبل تطبيق الهندسة العكسية. هذه العملية بسيطة، وهناك عدة أدوات يمكن أن تقوم بها. ولكن لسوء الحظ، هذه الأدوات لا تساعد في حل مشكلة الفهم لأنها تزيل مشاكل التحكم في البرنامج، لكنها لا تقوم بتغيير العدد الكلي لنقاط القرار. لذلك، لا تتغير القيمة المحددة، وتبقى الأوضاع المشابهة للشكل 10-9 بلا تغيير. تقوم الأدوات بتغيير موقع أو تكرر جزء من الشيفرة البرمجية بهدف القضاء على المشاكل، وفي العادة يكون الموقع النسبي لأجزاء الشيفرة البرمجية مناسباً ويعبّر عن معنى ما. لذلك يكون تغيير موقع أو تكرار شيفرة ما

من دون مراعاة المعنى قد يؤدي إلى التقليل من مستوى فهم البرنامج.

لهذه الأسباب، تم تعريف شكل جديد من عملية الصيانة الاستثنائية وتم تطبيقها على نظام البنوك: الاستعادة[41, 42]. بالتفصيل، تعمل الاستعادة على إعادة بناء البرمجية وفقاً للمعنى الذي تحمله الشيفرة البرمجية. من أجل فهم أفضل للشيفرة البرمجية، تقوم بما يأتي:

- تزيل البيانات منتهية الصلاحية، أي البيانات الموجودة ولكن لم يسبق أن تستخدم.
- تزيل البرامج المهملة، البرامج التي يتضمنها النظام وتستخدم البيانات الميتة، وبالتالي ينتج منها نتائج غير قابلة للاستخدام.
- تزيل الأوامر منتهية الصلاحية، أي الأوامر التي لم يسبق استخدامها.
- تحسن من قيمة البيانات والأسماء الإجرائية، وفقاً لتلك التي تكون غير مفيدة.

بعد التعرض للظروف السابقة، يجب أن يتم إعادة بناء الشيفرة البرمجية وفق الخطوات الآتية:

إزالة عمليات الشرط IF الإجرائية، لأن IF تحتل وحدتين من الشيفرة، كلٌ منها يحمل معنى متناسقاً ذاتياً، وبالتالي يمكن تنفيذها بشكل مستقل. للقضاء على عملية IF، يجب استدعاء كل وحدة في سياق مختلف، بناءً على حالة البرنامج.

الجدول (10 ـ 18)

مقايسات تظهر تأثير الإصلاح (Restoration)

الإجراءات	قبل الإصلاح	بعد الإصلاح
بيانات منتهية (Dead)	3.597	0
بيانات معرّفة بالكامل	9.000	5.403
برامج غير مستخدمة	1.252	0
برامج في المكتبات	1.891	639
تعليمات منتهية (Dead)	270.507	0
تعليمات كلية	1.502.734	1.232.227
IFS تجريبية	435.889	217.457

يلخّص الجدول 10-18 نتائج عملية الاستعادة. يظهر الشكل 10-10 تحسن الرسم البياني المتعلق بـ A0000 بعد إجراء الاستعادة، بعد تحسين هيكلية البرمجية، كان من الممكن إعادة تسمية 63 في المئة من البيانات والإجراءات، وإيجاد فهم أفضل لمجالاتهم. بناءً على ذلك، أصبح من الممكن تحسين الضعف في القاموس. بعد التحسين في فهم البرنامج، تقل مقدار المعرفة المضمنة. في الحقيقة، ازداد عدد الوظائف التي يمكن تتبعها في البرمجية إلى 852.

لإتمام العمل، أتاح تحسين عدد القيم المحددة للبرامج (لنسبة من البرامج مقداره 75 في المئة أصبح عددها بين 5 و15)، لكنها لا زالت فوق الحد الموصى به من قبل هندسة البرمجيات.

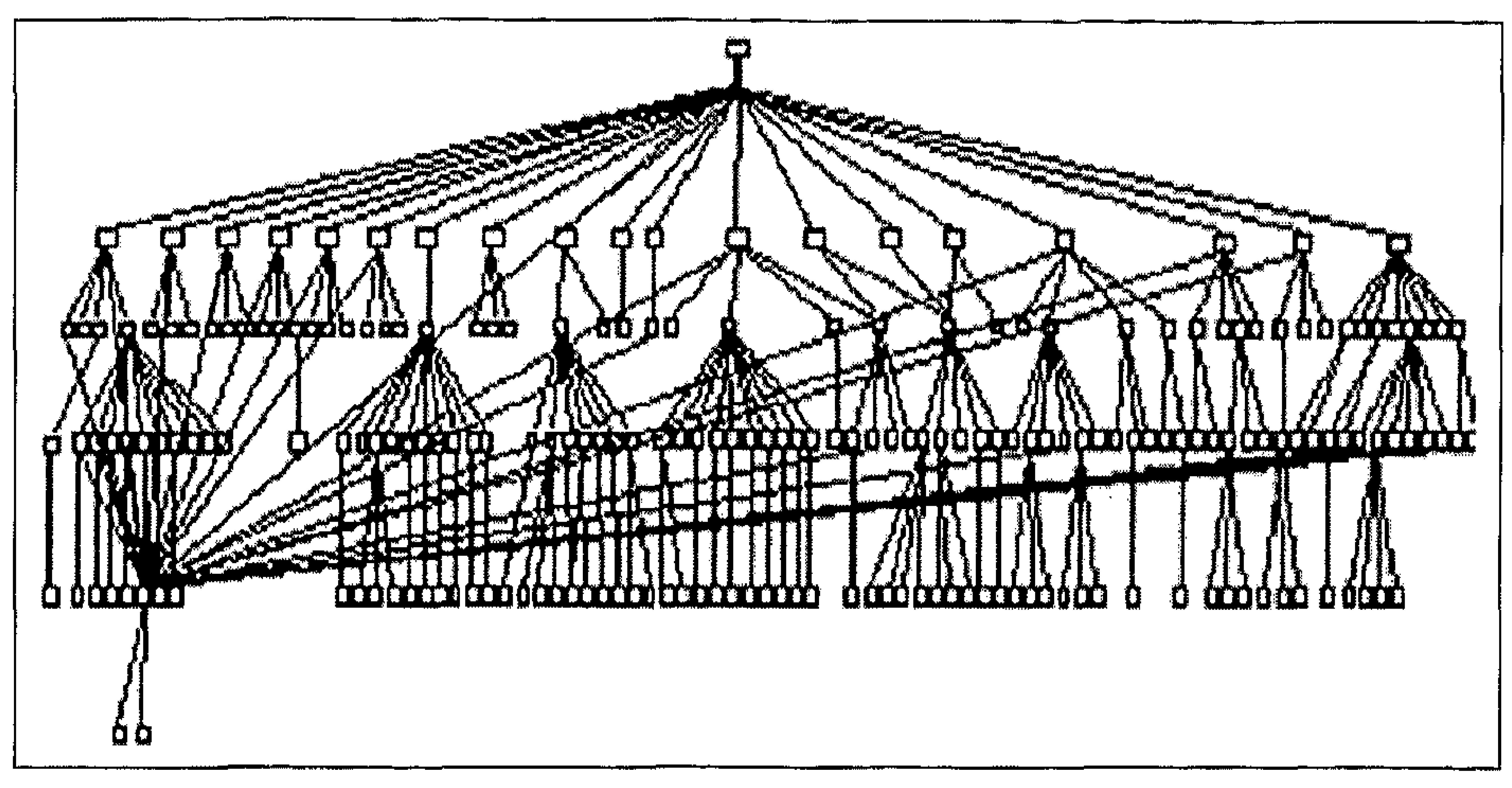
بعد إجراء الاستعادة، لا تزال البرامج تحتوي على نوع آخر من التقارن: الملفات الباثولوجيكية. يظهر الشكل 10-11 توزيع الملكية في نظام برمجي. على سبيل المثال، تشير إلى أن سبعة ملفات تعاني الباثولوجيكية في 40 برنامجاً. بقيت هذه التقارنات من دون تغيير حتى بعد إجراء الاسترجاع:

الدروس المستفادة هي:

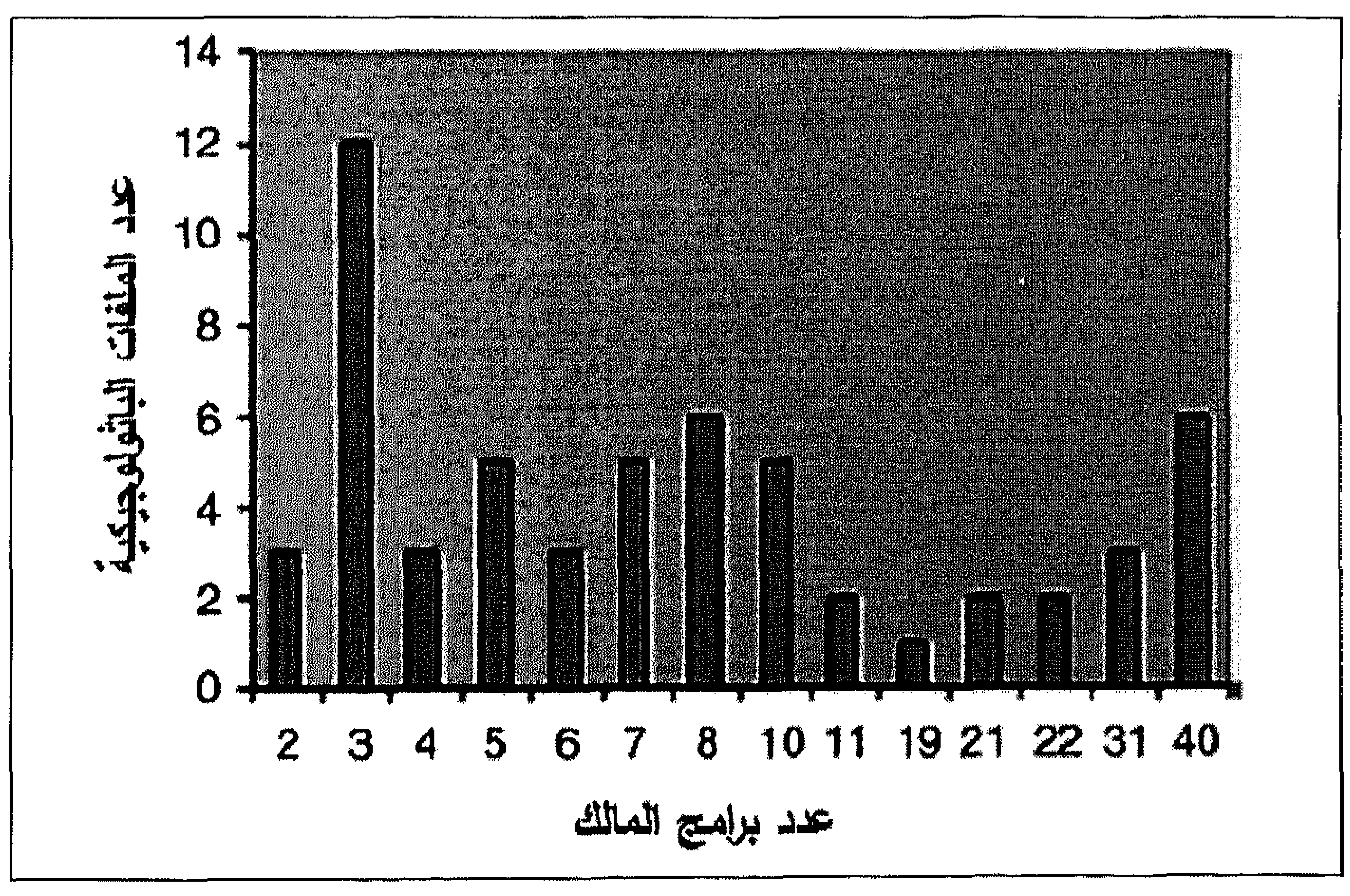
- تحسن عملية الاستعادة المعرفة المضمنة والضعف في القاموس والتقارن الجزئي. تبقى جودة الهيكلية والتصميم من دون تغيير. فمثلاً لا تقوم بتحسين المعلومات المخبأة في وحدات النظام.
- تم تحسين الجهد والكفاءة لعملية الصيانة.
- يتطلب الاسترجاع تقنيات دقيقة ولكن ليست رسمية، وبالتالي لا يمكن دعمها عن طريق الأدوات، لذلك تحتاج هذه العملية إلى جهد شخصي، وبالتالي تكون مكلفة.
- الخطورة الأخرى نتيجة المهارات المحددة التي يجب أن يمتلكها المطوّرون وتتطلبها العمليات
- لا يمكن وضع تعريف رسمي لإنهاء العمليات، فهو يمثل تطبيقاً مستمراً يحدد التحسن المستمر في الشيفرة أو في قاعدة البيانات، لكن هذا يعمل على زيادة خطورة رفع التكاليف.
- يمكن التقليل من خطورة ارتفاع التكاليف عن طريق تقسيم العمل إلى شرائح مناسبة لكل مكوّن سيتم استرجاعه. تتوقف العملية بعد نهاية كل

شريحة عمل، كلما كانت الشريحة تتطلب عملاً أكبر كانت جودة النظام المُستعاد أفضل. لذلك يجب أن يحدد مدير المشروع شرائح كبيرة من العمل للبرامج ذات الأهمية الكبيرة.

- يمكن التقليل من خطورة مهارات المطوّرين عن طريق تدريبهم تدريباً كافياً.



الشكل (10 ـ 10): رسم بياني يبيّن وحدة العلاقات المكوّنة للبرنامج A0000 بعد عملية الاسترجاع



الشكل (10 ـ 11): توزيع الملفات الباثولوجيكية

يتوجب الآن إجراء اختبارات على النظام البرمجي الذي تمّت صيانته على أسس اعتيادية، بقياس أعراض التقادم. عندما تتعدى هذه الأعراض حدوداً معيّنة، يجب إجراء عملية الاستعادة. يسمح ذلك بتجديد النظام عن طريق عمليات استعادة بسيطة، بتكلفة وخطورة منخفضتين.

10 ـ 5 ـ 5 إعادة التصميم

من الأهمية بمكان إعادة التصميم للقضاء على أعراض التقادم التي لم تستطيع عملية الاستعادة إزالتها. أدى تطبيقها إلى النتائج المبيّنة في الجدول 10-19. أظهرت القياسات التحليلية كيف أن إعادة التصميم مكّننا من التغلب على جميع أعراض تقادم النظام البرمجي.

تعتبر هذه العملية معيقة للعمل لأنها تتطلب معالجة جميع البيانات والإجراءات في نفس الوقت لجميع أجزاء النظام. لذا من الضروري إيقاف النظام البرمجي عن العمل خلال عملية إعادة التصميم، إضافة إلى ذلك، يجب إيقاف جميع إجراءات الصيانة الاعتيادية إلى أن تنتهي هذه العملية. من المستحيل تطبيق عمليات إعادة التصميم على البرمجية التي تستخدمها الشركة.

الجدول (10 ـ 19)

القياسات قبل وبعد إجراء عملية إعادة التصميم

	قبل التجديد	بعد التجديد
ملفات غير مستخدمة	7	0
ملفات مؤقتة	2	0
ملفات مهملة	2	0
ملفات باثولوجية	58	0
بيانات غير مكررة	3.825	4.061
بيانات مكررة ذات دلالة	1.578	632
بيانات حسابية مكررة	946	0
بيانات مستخدمة	5.403	4.693

حدّد المؤلف عملية إعادة التصميم التكرارية[44] للتغلب على هذه

المشكلة. تسمح هذه الطريقة بتجزيء النظام إلى مكوّنات، ويتم إعادة التصميم لكلٍّ منها بشكل مستقل عن الأخريات. يتم ضمان التوافق بين المكوّنات القديمة والمكوّنات التي تجري عليها عملية إعادة التصميم خلال تنفيذ هذه العملية، بذلك يمكن الاستمرار في استخدام النظام بشكل كلي والدخول إلى البيانات والوظائف القديمة التي تجري عليها عملية إعادة التصميم. الوقت الذي يستمر فيه النظام بالتوقف عن العمل يكون قليلاً، ويعتمد ذلك على حجم المكوّن الذي يتم إعادة هيكلته.

تم تنفيذ هذه العملية على نظام صناعي يدعم منتجين للمواد الكيميائية.

في البداية أظهرت العملية قابلية نقل إعادة التصميم، حتى بعد التدخل لإجراء التكرارات. خلال فترة إعادة التصميم، التي استغرقت 18 شهراً احتاجت إلى ثمانية وتسعين تدخلاً لإجراء المعالجة عليها، تم إيقاف 63 منها عن العمل لمدة تقل عن 10 أيام عمل، 28 لمدة تتراوح بين 11و15 يوماً، في حين أوقف 7 منها لمدة تتراوح بين 18 و28 يوماً.

الدروس المستفادة من هذه التجربة هي:

- هدف العملية هو تحسين الجودة التقنية للبناء الهندسي والتصميم، بالإضافة إلى تحديث القدرات الوظيفية والتقنية، عملية إعادة التصميم فاعلة في إزالة أعراض التقادم، التي لا يتم إزالتها عن طريق عمليات الصيانة الاستثنائية أخرى.
- يمكن إجراء عمليات إعادة التصميم المكررة من دون الحاجة إلى استرجاع النظام.
- تمكننا عملية إعادة التصميم من تحديث العمليات التي يزودها البرنامج، وإبطاء التناقص في القيمة التجارية.
- تتمثل الخطورة الرئيسة في هذه العملية في تحديد المكوّنات الواجد إعادة هيكلتها في كل فترة برمجية تكرارية: التقسيم غير المناسب قد يؤدي إلى زيادة فترات إعادة التصميم وبالتالي فترات أطول من الوقوف عن العمل خلال فترات الصيانة.
- يمكن التقليل من المخاطرة عن طريق استخدام تقنيات مناسبة[14].

10 ـ 5 ـ 6 الملخّص

يمكن تلخيص الأمر بتوضيح أن النتائج التي نحصل عليها خلال تنفيذ عمليات الصيانة الاستثنائية والدروس المستفادة من الاختبارات التجريبية، تم توضيحها في الفقرات السابقة هي الآن ملخصة. سيتم عرض الملخص على شكل أجوبة للأسئلة التي تم سردها سابقاً، التي تعبّر عن مدى فاعلية عملية صيانة الصيانة الاستثنائية. تم تنظيم التعليقات في الجدول 10-20.

الجدول (10 ـ 20)

توليفة محتويات الاختصاص

مكوّنات الاختصاص	عمليات الصيانة الاستثنائية البرمجية		
	الهندسة العكسية	الاستعادة	إعادة التصميم
متى تستخدم كل عملية	بعد تنفيذ مجموعة من أنشطة الصيانة التي تعدل الشيفرة البرمجية. عندما يكون إدخال المعرفة المضمنة في البرمجية من خلال الصيانة بطيئاً. عند تقادم النظام البرمجي، يجب أن يتم التحقق لتوصيف الإرث البرمجي وذلك قبل تطبيق هذه العملية	المعرفة المضمنة كبيرة أو قد يكون هناك معجم والوحدات متقارنة. عند تقادم النظام البرمجي، يجب أن يتم التحقق لتوصيف موجودات البرمجية قبل تطبيق هذه العملية	تحذف أعراض التقادم التي لم تكن عملية الصيانة الاستثنائية EM قادرة على الحد منها. عند تقادم النظام البرمجي، يجب أن يتم التحقق لتوصيف موجودات البرمجية قبل تطبيق هذه العملية
لماذا تستخدم عملية بعينها	يجب أن تستخدم لمراقبة تأثير التلوث الذي تحدده الصيانة. إذا نفذ ذلك في اللحظة الملائمة، فقد تتحسن المعرفة المضمّنة، تحديث التوثيق لحفظ التتبع بين مستويين غير تطبيقيين مختلفين في النظام.	تحسين الشمولية وتركيب النظام البرمجي بحيث تكون الصيانة العادية مجدية أكثر من حيث التكلفة والسرعة والموثوقية.	تتضمن أهداف العملية تحسين الجودة التقنية للهيكلية والتصميم التفصيلي مع تحديث القدرات الوظيفية والتقنية.
تكلفة ومنافع استخدام هذه الطريقة	التكلفة منخفضة لأن معظم الأنشطة مؤتمتة تقريباً؛ المنفعة المتحققة هي عملية التتبع التي تتيح للأفراد الذين يجرون الصيانة إجراء التغيرات بسرعة مستقبلاً.	تكاليف هذه العملية مرتفعة جداً؛ أما المنافع فتشمل تحسين القدرة على الصيانة التي تقلل من تكاليف الصيانة.	عملية مكلفة، لكنها تحسن القدرة على الصيانة وتحافظ على الاستثمارات في إنتاج البرمجيات لأنها تمد من عمر النظام.

يتبـــع

تابــع

المخاطر التي قد تعيق العملية وكيفية التخفيف منها	مخاطر استدامة الأدوات؛ تتطلب عملية التخفيف من المخاطر أن يوفر مزوّد الأدوات دليلاً تجريبياً أو أن يستنبط مدى صلاحيتها باستخدام التحقق من سلوك الأداة التي يتم إجراء اختبار الإجهاد عليها.	المخاطرة التي تحيق بهذه العملية هي شرط الإنهاء: وبغياب هذه المخاطرة قد تحدد فترات تنفيذ طويلة وجهداً كبيراً. وهذا قد يتلف توازن التكلفة ـ المنفعة. تتكون إجراءات التخفيف من تعريف نقطتي بداية: شريحة الجهد الذي يجب إنفاقه ومستوى جودة البرمجية الذي يجب الوصول إليه؛ عند الوصول إلى أي من نقطتي البداية هاتين، تتوقف عملية الاستعادة.	التحديد الفعال للمكوّنات التي يعاد تصميمها في كل فترة برمجية تكرارية، تتكون إجراءات التخفيف من المخاطرة الاستخدام الصحيح للعملية الموصوفة في المرجع 44.

10 ـ 6 الاستنتاجات

هدف هذا الفصل إلى تقديم المبادئ الأساسية المتوفرة في المادة النظرية للعديد من المؤلفين والباحثين، وللطلاب والباحثين المهتمين في الأبحاث التجريبية. يمثل هذا الفصل مقدمة إلى الاستقصاء التجريبي كطريقة للربط ما بين هندسة البرمجيات والعلم. يجمع الكثير من الخبرات المنشورة في مشروع SERLAB. لذلك تم توجيهه إلى القرّاء الذي ينوون أن يجعلوا الاستقصاء التجريبي جزءاً من عملهم. يمكن للقرّاء المهتمين أن يبدأوا باستعراض المراجع المقترحة والانطلاق منها إلى مراجع أكثر.

يشير هذا الفصل بأن عمليات الاستقصاء التجريبي في هندسة البرمجيات لها خصائص مشابهة لخصائص العلوم الإنسانية أكثر شبهاً بالعلوم الطبيعية، نتيجة لطبيعة العملية التي يوجهها الإنسان. هذه الطبيعة تتطلب ما يأتي:

- أخذ الاحتياطات خلال عمليتي التصميم والتنفيذ لعملية الاستقصاء التجريبي لتجنب تحديد تأثيرات غير موجودة.
- التكرارات من أجل تأكيد العلاقات المعقدة التي تنتج من الخصائص الإنسانية خلال انشاء البرمجية

- تقنيات وقدرات من أجل إعادة دقيقة للتجربة أو التخطيط لتغيير أحد العوامل أو أكثر، من أجل تحديد عائلة البرامج وتعميم النتائج.

بالإضافة إلى ذلك، يقوم هذا الفصل بتحليل حصيلة 10 أعوام من الخبرات في الأبحاث التي تم إجراؤها في مشروع SERLAB من أجل دعم وتحسين عملية نقل نتائج البحث والتقنيات المبتكرة في العمليات الصناعية. تفرض طريقة جمع البيانات خلال عمليات الإعادة التي تكون ضرورية للموازنة ما بين التكلفة والفوائد وتحدد المخاطر وتبيّن آلية إضفاء الطابع المؤسسي على الابتكار، على الرغم من أن ذلك ليس ضرورياً لتأكيد النظريات الأساسية. كما أشار هذا الفصل إلى أن كل نوع من أنواع عمليات الاختبار يساهم بطريقة مختلفة في تفضيل طلب ابتكار ما.

تبرز هذه الخبرة كيف أن عملية الاستقصاء التجريبي في هندسة البرمجيات، كما في العلوم الأخرى عامل فاعل في زيادة الكفاءات في أي من مجال من مجالات المعرفة. أخيراً، في هذا الفصل بقيت هذه الأمور من دون وجود حل لها:

- دراسة الطرق والأدوات للمبادىء المنهجية التي تجعل نتائج الاختبار دقيقة وذات موثوقية.
- تطوير أساليب العمل من أجل حفظ التجارب بحيث يمكن إعادتها بدقة أو بتغيير عوامل التجربة.
- استخراج البيانات الموصى بها وجمعها خلال عمليات الإعادة، التي تكون ضرورية ومناسبة لجعل الابتكار قيد الدراسة مقبولاً من قبل المشاركين.

المراجع

1. D. Altman [et al.]. «Statistical Guidelines for Contributors to Medical Journals.» *British Medical Journal*: vol. 286, 1983, pp.1489- 1493.

2. D. Altman. *Guidelines for Contributors, Statistics and Practice*, S. M. Gore and D. Altman, editors. London: British Medical Association, 1991.

3. D. Altman. «Statistical Reviewing for Medical Journals.» *Statistics in Medicine*: vol. 17, 1998, 2661-2674.

4. P. Ardimento [et al.]. «Innovation Diffusion through Empirical Studies.» paper presented at: *Proceedings of the 17th International Conference on Software Engineering and Knowledge Engineering (SEKE)*, Taipei, China, 2005, pp. 701-706.

5. P. Ardimento [et al.]. «Multiview framework for goal-oriented measurement plan design.» paper presented at: *Proceedings of 5th International Conference on Product Focused Software Process Improvement (PROFES)*, LNCS 3009, 2004, pp. 159-173.

6. P. Ardimento [et al.]. «Empirical investigation for building competences: A case for extraordinary maintenance.» paper presented at: *Proceedings of the 17th International conference on Software and Knowledge Engineering (SEKE)*, Taipei, China, 2005, pp. 695-700.

7. M. T. Baldassarre, D. Caivano, and G. Visaggio. «Comprehensibility and efficiency of multiview framework for measurement plan design.» paper presented at: *Proceedings of the International Symposium on Empirical Software Engineering (ISESE)*, Rome: Italy, 2003, pp. 89-99.

8. M. T. Baldassarre, D. Caivano, and G. Visaggio. «Noninvasive monitoring of a distributed maintenance process.» paper presented at: *Proceedings of IEEE Instrumentation and Measurement Technology Conference (IMTC 2006)*, Sorrento, Italy, 2006, pp. 1098-1103.

9. V. R. Basili and H. D. Rombach. «The TAME project towards Improvement-oriented software environments.» *IEEE Transactions on Software Engineering*: vol. 13, no. 12, 1987, pp. 1278-1296.

10. V. R. Basili. «Software development: A paradigm of the future.» paper presented at: *Proceedings of the International Computer Software and Applications Conference (COMPSAC)*, Orlando, FL, 1989, pp. 471-485.

11. V. R. Basili, G. Caldiera, and H. D. Rombach. Experience factory, J.-J. Marciniak, editor. *Encyclopedia of Software Engineering*. New York: Wiley, 1994, pp. 528-532.

12. V. R. Basili, F. Shull, and F. Lanubile. «Building knowledge through families of experiments.» *IEEE Transactions on Software Engineering*: vol. 25, no. 4, 1999, pp. 456-473.

13. C. Begg [et al.]. «Improving the Quality of Reporting of Randomized Trials (the CONSORT statement).» *Journal of the American Medical Association*: vol. 276, no. 8, 1996, pp. 637-639.

14. A. Bianchi, D. Caivano, and V. Marengo. «Iterative Reengineering of Legacy Systems.» *IEEE Transactions on Software Engineering*: vol. 29, no. 3, 2003, pp. 225-241.

15. E. J. Chifosky and J. H. Cross II. «Reverse Engineering and Design Recovery: A taxonomy.» *IEEE Software*: vol. 7, no. 1, 1990, pp. 13-17.

16. T. D. Cook and D. T. Campbell. *Quasi-Experimentation: Design and Analysis Issues for Filed Settings*. Boston, MA: Houghton Mifflin Company, 1979.

17. A. Endres and H. D. Rombach. *A Handbook of Software and System Engineering Empirical Observation Laws and Theories*. Reading, MA: Pearson Education Limited, Addison-Wesley, 2003.

18. H. Fukuda and Y. Ohashi. «A Guideline for Reporting Results of Statistical Analysis.» *Japanese Journal of Clinical Oncology*: vol. 27, no. 3, 1997, pp. 121-127.

19. M. J. Gradner and D. G. Altman. *Statistics with Confidence*. London: BMJ, 1989.

20. R. L. Glass. *Software Conflicts Essay on the Art and Science of Software Engineering*. New York: Yourdon Press, 1991.

21. M. M. Lehman and L. A. Belady. *Program Evolution: Processes of Software Change*. New York: Academic Press, 1985.

22. M. M. Lehman, D. E. Perry, and J. F. Ramil. «Implications of evolution metrics on software maintenance.» paper presented at: *Proceedings of the 1998 International Conference on Software Maintenance (ICSM'98)*, Maryland, 1998, pp. 208-217.

23. S. M. McGuigan. «The Use of Statistics in the British Journal of Psychiatry.» *British Journal of Psychiatry*: vol. 167, no. 5, 1995, pp. 683-688.

24. C. M. Judd, E. R. Smith, and L. H. Kidder. *Research Methods in Social Relations*. 6th ed. Orlando, FL: Harcourt Brace Jovanovich, 1991.

25. N. Juristo and A. M. Moreno. *Basics of Software Engineering Experimentation*. Boston, MA: Kluwer Academic Publishers, 2001.

26. E. Kamsties and C. Lott. «An Empirical Evaluation of there Defect Detection Techniques. Technical report ISERN 95-02, Department of Computer Science, University of Kaiserslautern, May 1995.

27. B. A. Kitchenham [et al.]. «Toward an Ontology of Software Maintenance.» *Journal of Software Maintenance: Research and Practice*: vol. 11, no. 6, 1999, pp. 365-389.

28. B. A. Kitchenham [et al.]. «Preliminary Guidelines for Empirical Research in Software Engineering.» *IEEE Transactions on Software Engineering*: vol. 28, no. 8, 2002, pp. 721-734.

29. J. Lewis [et al.]. «An empirical study of the object-oriented paradigm and software reuse.» paper presented at: *Proceedings of the Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA)*, 1991, pp. 184-196.

30. C. M. Lott and H. D. Rombach. «Repeatable Software Engineering Experiments for Comparing Defect-Detection Techniques.» *Empirical Software Engineering*: vol. 1, no. 3, 1996, pp. 241-277.

31. C. Nachmias and D. Nachmias. *Research Methods in the Social Sciences.* London: Edward Arnold, 1981.

32. D. L. Parnas. «On the Criteria to be used in Decomposing System in Modules.» *Communications of the ACM*: vol. 15, no. 12, 1972, pp. 1053-1058.

33. S. L. Pflegger. «Soup or Art? The Role of Evidential Force in Empirical Software Engineering.» *IEEE Software*: vol. 20, no. 1, 2005, pp. 66-73.

34. A. M. Porter. «Measure of Correlation and Regression in Three Medical Journals.» *Journal of the Royal Society of Medicine*: vol. 92, no. 3, 1999, pp. 123-128.

35. R. Rosenthal. «Replication in Behavioral Research.» *Replication Research in the Social Sciences.* J. W. Neuliep, editor. Thousand Oaks, CA: Sage Publications, 1991.

36. I. Rus and M. Lindvall. «Knowledge Management in Software Engineering.» *IEEE Software*: vol. 19, no. 3, 2002, pp. 26-38.

37. H. S. Sacks [et al.]. «Meta-Analyses of Randomized Control Trials.» *The New England Journal of Medicine*: vol. 316, no. 8, 1987, pp. 312-455.

38. D. A. Scanlan. «Structured Flowcharts Outperform Pseudocode: An Experimental Comparison.» *IEEE Software*: vol. 6, no. 5, 1989, pp. 28-36.

39. B. Schneiderman [et al.]. «Experimental Investigation of the Utility of Detailed Flowcharts in Programming.» *Communications of the ACM*: vol. 20, no. 6, 1977, pp. 373-381.

40. G. Visaggio. «Assessment of a Renewal Process Experimented in the Field.» *The Journal of Systems and Software*: vol. 45, no. 1, 1999, pp. 3-17.

41. G. Visaggio. «Value-Based Decision Model for Renewal Processes in Software Maintenance.» *Annals of Software Engineering*: vol. 9, nos. (1-4), 2000, pp. 215-233.

42. G. Visaggio. «Ageing of a Data-Intensive Legacy System: Symptoms and Remedies.» *Journal of Software Maintenance and Evolution: Research and Practice*: vol. 13, no. 5, 2001, pp. 281-308.

43. L. Wilkinson and Task Force on Statistical Inference. «Statistical Methods in Psychology Journals: Guidelines and Explanations.» *American Psychologist*: vol. 54, no. 8, 1999, pp. 594-604. (< http://www.apa.org/journals/amp/amp548594.html >).

44. C. Wohlin [et al.]. *Experimentation in Software Engineering: An Introduction*. Boston, MA: Kluwer Academic Publishers, 2000.

11

أساسيات المنهجات السريعة

البرتو سيليتي (Alberto Sillitti)
وجانكارلو سوتشي (Giancarlo Sussi)

11 ـ 1 مقدمة

المنهجيات السريعة هي مجموعة من تقنيات التطوير المصممة لعَنوَنة بعض مشكلات تطوير البرمجيات الحديثة (أي المشاريع التي تتجاوز الموازنة أو تتجاوز جدول التنفيذ). لا يبدو أن مثل هذه المنهجيات مفيدة لأي نوع من مشاريع البرمجيات أو أن تكون حلاً للحد من تكاليف المنتج وزيادة جودته. لكن، في سياقات محددة ولمشكلات محددة، تساعد المنهجيات السريعة المطوّرين في التركيز على أهداف العملاء وتسليمهم المنتج الصحيح من دون هدر الوقت والجهد في أنشطة ليست ذات قيمة للعميل.

تتطلب منهجيات تطوير البرمجيات التقليدية (أي منهجية الشلال (Waterfall) والحلزونية (Spiral) والتكرارية (Iterative) وغيرها) معرفة عميقة في مجال التطبيق وفي احتياجات العميل الفعلية (بما في ذلك المستخدم النهائي للتطبيق). لكن، هذه المعرفة نادراً ما تكون متوافرة حتى في الحالات التي يطلب فيها العميل إجراء تغيير في أثناء مرحلة التطوير. لسوء الحظ، تتميّز عملية تطوير البرمجيات بعدم اليقين (Uncertainty) وعدم القدرة على التراجع (Irreversibility)[10, 5]؛ لذا لن يكون تخطيط كل شيء مسبقاً مفيداً في العديد من مجالات التطبيقات.

عدم اليقين (Uncertainty) يعني أن المتطلبات غير ثابتة والعميل غير قادر

على تحديدها بطريقة كاملة ومتماسكة. غالباً ما لا يكون العملاء قادرين على تحديد الوظائف الرئيسة المطلوبة ويغيّرون رأيهم بصورة متكررة.

عدم القدرة على التراجع (Irreversibility) تعني أنه حتى لو كانت البرمجية معنوية، فإنه لا يمكن تغيير بعض القرارات من دون التأثير بعمق في جدول تنفيذ المنتج والموازنة. هذا الأمر مفهوم في علوم أخرى كالهندسة المدنية. إذ يكون العميل مدركاً أنه لا يمكنه طلب تغيير شكل البناء في مرحلة إضافة السقف. لكن في مجال البرمجيات، غالباً ما لا يدرك العملاء أن إجراء تعديلات معيّنة له تأثير قابل للمقارنة.

أما العواقب الرئيسة لعدم اليقين وعدم القدرة على التراجع فهي:

1. المعرفة التامة اللازمة لبناء نظام لا تكون متوافرة دائماً و/أو تكون عرضة للتغيير في أثناء التطوير.

2. وضع افتراضات بتجريب بعض الحلول، ومن ثم إلقائها جانباً إذا لم تكن مطابقة؛ وهذا لا يُطبَق دائماً (هدر للوقت والمال).

تحاول تقنيات تطوير البرمجيات التقليدية الحد من عدم اليقين وعدم القدرة على التراجع من خلال وضع خطط مفصلة. فهم يحاولون تحديد كل شيء في البداية وذلك لتجنب التغيرات المكلفة التي قد تُطلب في مراحل متقدمة من المشروع. لكن في مجالات تطبيقات معيّنة ولبعض المشكلات المحددة، لا تعمل الخطط ببساطة أو قد لا تكون ذات فعالية. وهذا صحيح بغضّ النظر عن كفاءة العاملين في المشروع.

حتى لو كان يفترض أن تساعد الخطط الشركات، يعترف العديد من مدراء المشاريع أنه في أنواع كثيرة من المشاريع لا يمكنهم متابعة المشروع بسبب احتياجات السوق، وغالباً ما ينجحون.

تقرّ المنهجيات السريعة بصعوبة تعريف خطط تفصيلية عند بدء مشروع ويتضمن دعم التغييرات التي تطرأ على عملية التطوير. بهذه الطريقة، يتم جمع المتطلبات ومناقشتها مع العميل لكامل فترة المشروع بحسب الانطباعات والآراء التي يزود العميل بها فريق التطوير. تتم عملية التطوير بطريقة الفترات البرمجية التكرارية (Iteratively) ومن ثم يتم تسليم المنتج للعميل بعد كل فترة تكرارية لتقييمه (وتستغرق الفترة من أسبوعين إلى شهرين في منهجية XP).

تركّز المنهجيات السريعة على الناتج النهائي لعملية التطوير (أي الشيفرة البرمجية) وعلى القيمة التي توفّرها للعميل. المعطيات الإضافية (أي وثائق التصميم والتوثيق الكثير، إلخ) تعتبر هدراً لأن إنجازها يتطلب وقتاً وتصبح غير مفيدة ومضللة إذا لم تنجز بالطريقة الملائمة. إن الصيانة الضعيفة للتوثيق هو أمر شائع في حال حدوث تأخير في التطوير. في مثل هذه الحالات، يبذل كل الجهد في البرمجة واختبار الشيفرة البرمجية من دون تحديث التوثيق بصورة ملائمة.

بما إن هذا السيناريو مشترك تماماً، فإن الفكرة الأساسية هي التقليل من الزمن المستغرق لإنجاز التوثيق غير الضروري واستخدام الأدوات المؤتمتة لإعادة التصميم لإنتاج المنتج عند الحاجة.

غالباً ما يُساء تفسير المنهجيات السريعة. على سبيل المثال، يقول البعض إنهم يطبقون المنهجيات السريعة بسبب ما يأتي:

1. أنهم ينتقلون فوراً إلى البرمجة من دون كتابة وثائق التحليل والتصميم.

2. أنهم لا يقومون بكتابة التوثيق.

3. أنهم يقللون من الوقت المستغرق في الاجتماعات.

لكن، هذه ليست منهجية سريعة، بل برمجة بطريقة cowboy coding(*). توفّر المنهجيات السريعة طريقة لتنظيم تطوير البرمجيات من نواح تتجاوز الخطط، لكن لا يزال ذلك ضمن عملية صارمة يجب اتباعها. غالباً، يشكّل استخدام المنهجيات السريعة تحدياً أكبر من تطبيق أساليب تطوير البرمجيات التقليدية بما إن المنهجيات السريعة تتطلب مستوى أعلى من الالتزام ومهارات أكبر وغير ذلك. إن تطبيق وإدارة هذه المنهجيات أمر في غاية الصعوبة لكنها مصممة للتغلب على التحديات التي يفرضها سوق البرمجيات الحديثة (أي توفير هذه البرمجيات في السوق خلال فترة قصيرة ومتطلبات الجودة العالية وغير ذلك).

(*) البرمجة بطريقة cowboy هي مصطلح يُستخدم لوصف تطوير البرمجيات، يكون للمبرمجين سيطرة على عملية التطوير، بما في ذلك التحكم بجدول المشروع ولغة البرمجة والخوارزميات والأدوات وأطر العمل وأسلوب البرمجة. قد تتم البرمجة هنا من قبل مبرمج واحد أو مجموعة من المبرمجين. تستخدم هذه الطريقة عندما يكون هناك مشاركة قليلة من المعنيين بالأعمال أو عند وجود إدارة تحكم الجوانب غير المتعلقة بتطوير المشروع فقط، كالأهداف العريضة والجداول الزمنية ونطاق العمل (المترجم).

إضافة إلى ذلك، تتطلب المنهجيات السريعة سرعة أكبر مما يتطلبه الهيكل التنظيمي لشركة (أي أنواعاً جديدة من العقود وقرارات أكثر يترك أمرها لفريق العمل. . . إلخ) وأشخاصاً أكثر مرونة وذوي مهارات عديدة والقدرة على أداء العديد من الأدوار ضمن فريق التطوير. وبناء على ذلك، لا تكون المنهجيات السريعة ملائمة للجميع وهي لا تلائم جميع المشاريع البرمجية.

يحلل هذا الفصل المفاهيم الأساسية للتطوير السريع والصعوبات التي تعترض التنفيذ الفعال. حتى لو كان هناك العديد من منهجيات التطوير السريعة (البرمجة القصوى XP، Scrum، أسلوب تطوير النظم الديناميكية (DSDM)، Crystal، النمذجة السريعة وغيرها)، فإننا نركز على الإصدار الأول من البرمجة القصوى XP[2] وذلك لأنها الأكثر شهرة.

11 ـ 2 المنهجيات السريعة

المنهجيات السريعة هي عائلة من تقنيات التطوير المصممة لتوفير المنتج ضمن الوقت والموازنة المحددين بحيث يكون ذا جودة عالية ويحوز على رضا العميل[1, 7]. تتضمن هذه العائلة العديد من الأساليب المختلفة، وأشهرها:

- البرمجة القصوى (eXtreme Programming (XP[2, 4].
- 5.Scrum[16]
- أسلوب تطوير النظم الديناميكية (DSDM)[19]
- تطوير البرمجيات التكيّفية (ASD)[8]
- العائلة الكريستالية[6]

إن هدف هذه الأساليب هو تسليم المنتجات بشكل أسرع وبجودة عالية، بحيث تحوز رضا العملاء من خلال تطبيق مبادئ الإنتاج باتباع منهجيةLean على تطوير البرمجيات[15].

طوّرت مبادئ الإنتاج باتباع منهجية Lean[22] في أثناء عقد الخمسينيات من القرن الماضي من قبل شركة تويوتا[13]. تتضمن هذه المبادئ العديد من الممارسات التي تعتبر اليوم جزءاً من معظم العمليات التصنيعية كعمليات «في الوقت المحدد just-in-time» وإدارة الجودة الشاملة وتحسين العمليات المستمر.

أما المبدأ الأساسي في الإنتاج باتباع منهجية Lean فهو تحديد الهدر وإزالته (muda باللغة اليابانية) ـ أي أن الهدر هو أي شيء لا يضيف قيمة للعميل في المنتج النهائي.

بما إن المنهجيات السريعة هي تطبيق لمنهجيةLean في حقل صناعة البرمجيات، فإنها تركز على ما يأتي:

1. توفير قيمة للعميل.
2. إرضاء العميل.

إن توفير قيمة للعميل يتضمن أن يعمل فريق التطوير على إنتاج ما يوفر قيمة والحد من الأمور الأخرى إلى الحد الأدنى. تشدد المنهجيات السريعة على إنتاج ميزات مفيدة فقط (من وجهة نظر العميل) وتسليمها للعميل. إن إنتاج أي شيء آخر غير متطلب يعتبر خطأً.

تحديداً، إن إضافة ميزة غير مطلوبة لا تتطلب جهداً فحسب، بل إنها تضيف شيفرة برمجية إضافية، التي قد تتضمن أخطاءً وتجعل الشيفرة البرمجية أكبر وأكثر تعقيداً ما يصعب من صيانتها وتصحيحيها وتحسينها.

للحد من الهدر، تدعي المنهجيات السريعة[1] أنها:

- تكيُّفية بدلاً من أن تكون توقُّعية.
- أنها قائمة على الأشخاص بدلاً من كونها قائمة على العمليات.

لضمان رضا العملاء، يتطلب الأمر وجود تعاون وثيق بين فريق التطوير والعميل. لذا:

- تكون المتطلبات محددة بالكامل ومفهومة بالشكل الصحيح.
- تعكس المنتجات النهائية ما يرغب به العميل تحديداً، لا أكثر ولا أقل.

11 ـ 3 بيان منهجية التطوير السريع

يلخص بيان منهجية التطوير السريع (http://www.agilemanifesto.org/) الخلفية الأساسية والمشتركة لجميع المنهجيات السريعة. تعرّف هذه الوثيقة هدف المنهجيات السريعة وهي النقطة المرجعية لمجتمع البرمجة كاملاً.

تركز المنهجيات السريعة على العامل البشري في عملية التطوير وأهمية

التواصل المباشر وجهاً لوجه بين المشاركين والمساهمين في المشروع، وقيمة البساطة المتصورة كالحد من الهدر وتحسين مستمر في العملية، كما هو التحول في صناعة البرمجيات لإدارة الجودة الشاملة[15].

تعرّف المنهجيات السريعة كمجموعة من أساليب التطوير التي تشترك في القيم الأربع الآتية:

1. الأفراد وتفاعلهم يفوقون أهمية العمليات والأدوات: يركّز بيان المنهجيات السريعة AM على التعاون بين المطوّرين والدور الإنساني في العملية وفي المؤسسة خلافاً لعمليات وأدوات التطوير المؤسسي[1].

2. تعاون العميل يفوق أهمية العقود: يعطي بيان المنهجيات السريعة أهمية للتعاون بين المطوّرين والعملاء أكثر من الأهمية التي توليها لتحديد عقود مفصلة وواضحة[1]. إن التواصل غير الرسمي بين فريق العمل والعميل قد يحل محل العديد من الوثائق المكتوبة، حتى العقود التفصيلية.

3. برمجية عاملة تفوق أهمية التوثيق: إن العمليات الإضافية أو التوثيق هو أحد أهم مصادر الهدر في تطوير البرمجيات؛ إذ تستهلك الأعمال الورقية الموارد وتبطئ من زمن الاستجابة وتخفي مشكلات الجودة وقد تتعرض للضياع أو تضعف أهميتها أو قد تصبح باطلة لتقادمها. عندما يكون العمل الورقي مطلوباً، من الضروري أن يكون قصيراً وذا مستوى متقدم وتنفيذه من دون الاتصال بالإنترنت. يركّز بيان المنهجيات السريعة على فهم المنتج من خلال التعاون مع العميل وتسليم برمجية عاملة، وبذا تقل كمية الوثائق المطلوبة[9].

4. الاستجابة للتغيير تفوق أهمية اتباع خطة: «السرعة هي القدرة على الإنشاء والاستجابة للتغير بهدف تحقيق ربح في بيئة أعمال مضطربة»[9]. أما التغيير فهو فرص لمساعدة العملاء على تحديد الاضطراب في السوق بدلاً من تحديد مشكلات التطوير.

تشير أول قيمتين إلى إدارة الموارد البشرية، في حين أن القيم الأخيرة في القائمة السابقة تشير إلى إدارة العمليات.

من هذه القيم، هناك بعض المبادئ المشتقة المشتركة بين جميع بيانات المنهجيات السريعة. أما المبادئ الرئيسة المدرجة في بيان المنهجية السريعة فهي ما يأتي:

● الأولوية الكبرى هي إرضاء العميل من خلال تسليم برمجية ذات قيمة مبكراً وباستمرار: يجب على مطوّري البرمجيات تسليم العميل المنتج تدريجياً بدءاً من المتطلبات الأكثر أهمية.

● الترحيب بتغيير المتطلبات، حتى لو كان ذلك في مراحل متأخرة من عملية التطوير. تسخّر العمليات السريعة التغيير لصالح العميل: التغيير ليس بالأمر السيئ، لكن حالة طبيعية في مشاريع البرمجيات. لا يمكن منع التغيير الذي لا يمكن توقعه دائماً، وعليه فإن عملية التطوير يجب أن تستوعب التغيير ولا تحاربه.

● يجب أن يعمل المختصون بالأعمال والمطوّرون جنباً إلى جنب يومياً خلال المشروع: إن التعاون الوثيق بين فريق العمل والعميل هو طريقة للحد من المخاطر التي قد تعترض المشروع بما أنه يتم التحقق من صحة تفسير احتياجات العميل في كل خطوة من خطوات المشروع. بهذه الطريقة، يتم الحد من حالات إعادة العمل الناجمة عن سوء فهم المتطلبات، ويكون المطوّرون على معرفة دائمة للاستمرار في الطريق الصحيح.

● تسليم برمجية عاملة بشكل متكرر، قد تتراوح فترات التسليم من عدة أسابيع إلى عدة أشهر مع أفضلية لمقياس زمني أقصر: إن البرمجية القابلة للعمل هي الشيء القيم الذي يقدره العميل حتى لو لم تكن مكتملة. إن تسليم مجموعة من الوظائف المطلوب توفيرها في البرمجية دورياً وليس مجرد نماذج يعطي العميل القدرة على استخدام المنتج مبكراً. إضافة إلى ذلك، يزيد ذلك من وضوح حالة ووضع المشروع.

● بناء المشاريع بالاستعانة بأفراد لديهم حافز للعمل: لا ينطبق بيان المنهجيات السريعة على الجميع؛ إذ إن المطوّرين الذين يملكون المهارات والحافر للعمل القادرين على العمل ضمن فريق هم عامل أساسي للنجاح.

● وَفِّر لهم بيئة عمل ملائمة والدعم الذي يحتاجونه؛ ثِقْ بهم لتضمن إنجاز العمل: يجب على المدراء عدم التدخل في اجراءات فريق التطوير. إن الثقة والدعم اللذين توفرهما الإدارة لهي طريقة جيدة لدعم فريق عمل ذي مهارات ودوافع للعمل.

• أكثر الوسائل فعالية وكفاءة لنقل المعلومات لفريق التطوير هي النقاش المباشر وجهاً لوجه: حتى لو كان هناك العديد من طرق التواصل، إلا أن التواصل وجهاً لوجه هو أكثر الطرق فعالية لمنع حدوث سوء فهم وتقصير الوقت المستنفد لتبادل المعلومات.

• البرمجية العاملة هي المقياس الأوّلي على تقدم العمل: النتيجة المهمة الوحيدة للعميل هي تسليم برمجية عاملة. فلا أهمية إذا كان أنتج الفريق وثائق تصميم جميلة في حين أن المنتج لا يعمل أو أنه لا يلبّي احتياجات العميل. يقيس العميل مدى التقدم في المشروع بقياس عدد الوظائف المسلمة التي تعمل فعلياً.

• ترفع العمليات السريعة من التطوير المستدام. يجب أن يكون كفلاء المشروع والمطوّرون والمستخدمون قادرين على الحفاظ على وتيرة إلى أجل غير مسمى: يجب أن يتم تنفيذ تطوير البرمجيات من خلال تعاون ثابت ومستمر بين جميع أطراف المشروع وأصحاب المصلحة. إضافة إلى ذلك، يجب أن يكون جهد فريق التطوير ثابتاً مع الوقت، ومحاولة تجنب فترات التوتر والإجهاد غير الدائمة على المدى البعيد، ما يؤثر في دوافع التحفيز وإنتاجية فريق العمل.

• إن الانتباه المستمر للامتياز التقني والتصميم الجيد يحسّنان من سرعة التنفيذ: المطوّرون المتميزون ينتجون برمجيات ذات مستويات متقدمة. المطوّرون المتميزون قادرون على إنشاء نظم يمكن تحسينها وصيانتها بما يقلل من الوقت والجهد اللازمين.

• البساطة ـ فن إنجاز أكبر كمية من العمل غير المنجز ـ أمر ضروري: إن تحديد الشيفرة البرمجية غير المتطلبة هو طريقة لتحسين كفاءة فريق التطوير. إن الشيفرة البرمجية الأكثر بساطة أسهل للكتابة والفهم والصيانة. إضافة إلى ذلك، البرمجة الأقل تعني احتمالية أقل للأخطاء، وتتطلب اختباراً أقل.

• تنشأ أفضل الهيكليات والمتطلبات والتصاميم من فرق العمل ذاتية التنظيم: فريق العمل هو الوحيد المسؤول عن المنتج كاملاً. ليس هناك تنافس بين فرق العمل التي تركز على أنشطة محددة (أي التصميم والتنفيذ). يعمل جميع أعضاء الفريق لتحقيق هدف مشترك، وهو بالتحديد تحقيق رضا العميل.

● يعكس الفريق كيفية أن يصبح أكثر كفاءة على فترات منتظمة، ومن ثم يعمل على ضبط سلوكه بناءً على ذلك: عملية التطوير ليست ثابتة. يجب على فرق العمل أن تحدد مجالات التحسين باستمرار وذلك باختبار وتقييم تقنيات جديدة.

تنفّذ المبادئ المدرجة ويتم التشديد عليها بطرق مختلفة في العديد من بيانات المنهجيات السريعة المتوافرة. لكن، التركيز على احتياجات العميل والتحسين المستمر هما المبدآن الأساسيان.

إن أفكار العميل التي تقود الإنتاج والتحسين المستمر في العملية ليست بدعاً. فهي المفاهيم الأساسية لمبادئ الإنتاج باتباع منهجية Lean[13] [22, 13] ولمبدأ الإنتاج «في الوقت المناسب»[*] الذي استخدم في شركة تويوتا عام 1960 في صناعة السيارات. إن بيانات المنهجيات السريعة هي تنفيذ لهذه المفاهيم في صناعة البرمجيات[15].

11 ـ 4 البرمجة القصوى XP

ربما تكون البرمجة القصوى هي أكثر المنهجيات السريعة شهرة. في الوقت الحاضر، هناك إصداران من XP معرفان في طبعتين من كتاب Beck[2] [4, 2]. لكن في هذا الفصل سنأخذ بالاعتبار الإصدار الأول، وذلك لأنه الأكثر شعبية والأكثر قبولاً في مجتمع البرمجة. أما الثاني فهو الاقتراح الأحدث الذي قدمه كينت بيك (Kent Beck)، حيث يقترح تعديلات وتحسينات مهمة. لكن، لا يزال مجتمع البرمجة يناقشه، لذا فهو ليس متقبلاً على نطاق واسع. إضافة إلى ذلك، الإصدار الأول أكثر بساطة ويجب أن يكون نقطة البدء لأولئك الذي يسعون إلى بيانات المنهجيات السريعة للمرة الأولى.

تعرّف ممارسات XP كوصف لعملية ناجحة يتبعها فريق C3[2] الذي طور نظامَ رواتبٍ ضخماً لشركة كرايسلر. قاد كينت بيك فريق C3 بنجاح، حيث تمكنوا من تسليم نظام عامل بعد سنتين، في حين كان فريق العمل الأول غير قادر على تسليم أي شيء في أربع سنوات عمل.

(*) استراتيجية لإدارة المخزون، تسعى إلى تحسين العائد على الاستثمار من الأعمال والجودة والفاعلية عن طريق تخفيض المخزون المستخدم في العمليات وما يرتبط به من تكاليف. وهذا يعني أن المكونات والعناصر اللازمة لعملية الإنتاج قد تصل في الوقت المناسب ليختار العمال منها واستخدامها. وهذا يساعد في الحد من التخزين والجرد وتكاليفهما إذ إن المخزون يصل في الوقت المناسب (المترجم).

تعرف منهجية XP تسلسل مبادئ وممارسات التطوير بالتفصيل. لكن، لم يعرف هذا التسلسل بصورة مباشرة، لكنه اشتق من القيم والمتحكمات التي تعتبر أساس المنهجية (الشكل 11-1).

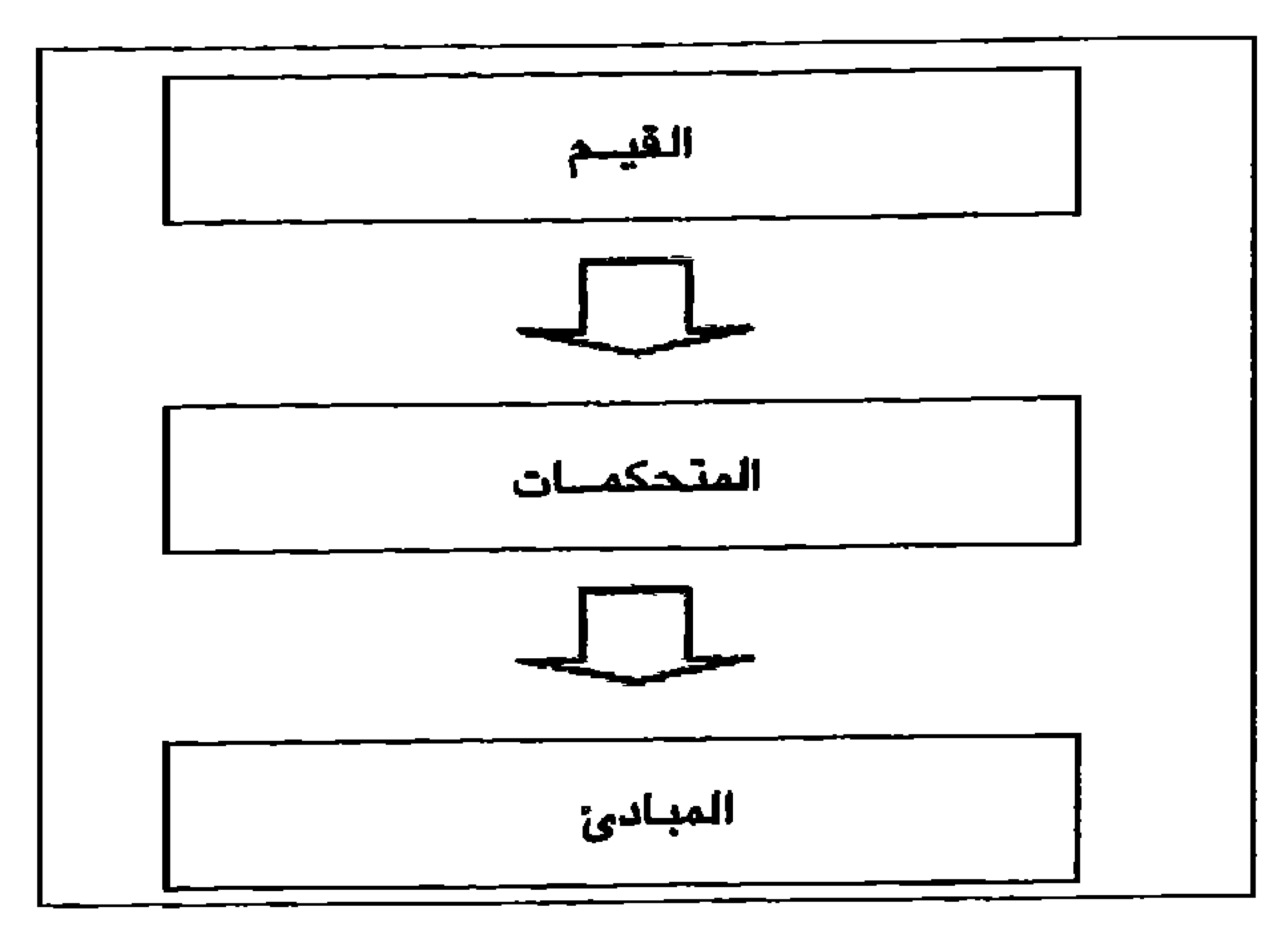
أما **قيم** منهجية XP فهي كما يأتي:

● **البساطة**: يجب أن يكون النظام بسيطاً ما أمكن لتلبية احتياجات العميل. لكن ليس أكثر بساطة. تنفيذ الميزات المطلوبة لكن يجب عدم شمول ميزات تدعم المتطلبات المستقبلية التي قد تصبح متطلبات حقيقية.

● **التواصل**: يجب أن يحسّن كل شيء من التواصل بين العملاء والمطوّرين وبين المطوّرين أنفسهم، وبين الشيفرة البرمجية المصدرية والقارئ. إن التواصل الفاعل يقلل من سوء الفهم والحاجة إلى بذل الوقت والجهد في التوثيق الرسمي والمكتوب.

● **الانطباعات وردود الأفعال**: يجب أن يحصل المطوّرون على انطباعات العملاء وردود أفعالهم بسرعة على كافة المستويات. يجب أن يحقق العملاء والمدراء والمطوّرون فهماً مشتركاً للهدف من المشروع وعن الوضع الحالي للمشروع وما يحتاجه العملاء أولاً، وما هي أولوياتهم، وما يستطيع المبرمجون عمله وفي أي وقت. وهذا يتحقق بوضوح عن طريق التواصل. يجب أن يكون هناك تغذية استرجاعية فورية من العمل الذي ينفذه القائمون على المشروع، أي من الشيفرة البرمجية التي يتم إنتاجها. يجب أن تنفق نسبة كبيرة من جهود البرمجة (حوالى 50 في المئة) في تطوير الاختبارات المؤتمتة. بهذه الطريقة تكون جودة النظام المطوّر عالية على نحو منطقي.

● **الشجاعة**: يجب أن يتحلى كل شخص ذي علاقة بالمشروع بالشجاعة (والحق) لعرض وضعه في المشروع. يجب أن يكون لدى الجميع الشجاعة والفكر المنفتح والسماح للجميع بالتحقق من عمله وتعديله. يجب أن لا يتم عرض التغيير بترهيب ويجب أن تكون لدى المطوّرين شجاعة للعثور على حلول أفضل وتعديل البرمجة عندما يكون الأمر لازماً ومجدياً. إن استخدام الأدوات الصحيحة وحزم الاختبار الشاملة يُمكّن البرمجة وتجربة حلول جديدة من دون خوف. وهذا يعني أنه من السهل اختبار ما إذا كان التعديل ينتج أخطاءً، ومن السهل الرجوع إلى الإصدار الأسبق القابل للعمل، إن لزم الأمر.



الشكل (11 ـ 1): القيم والمتحكّمات والمبادئ في منهجية XP

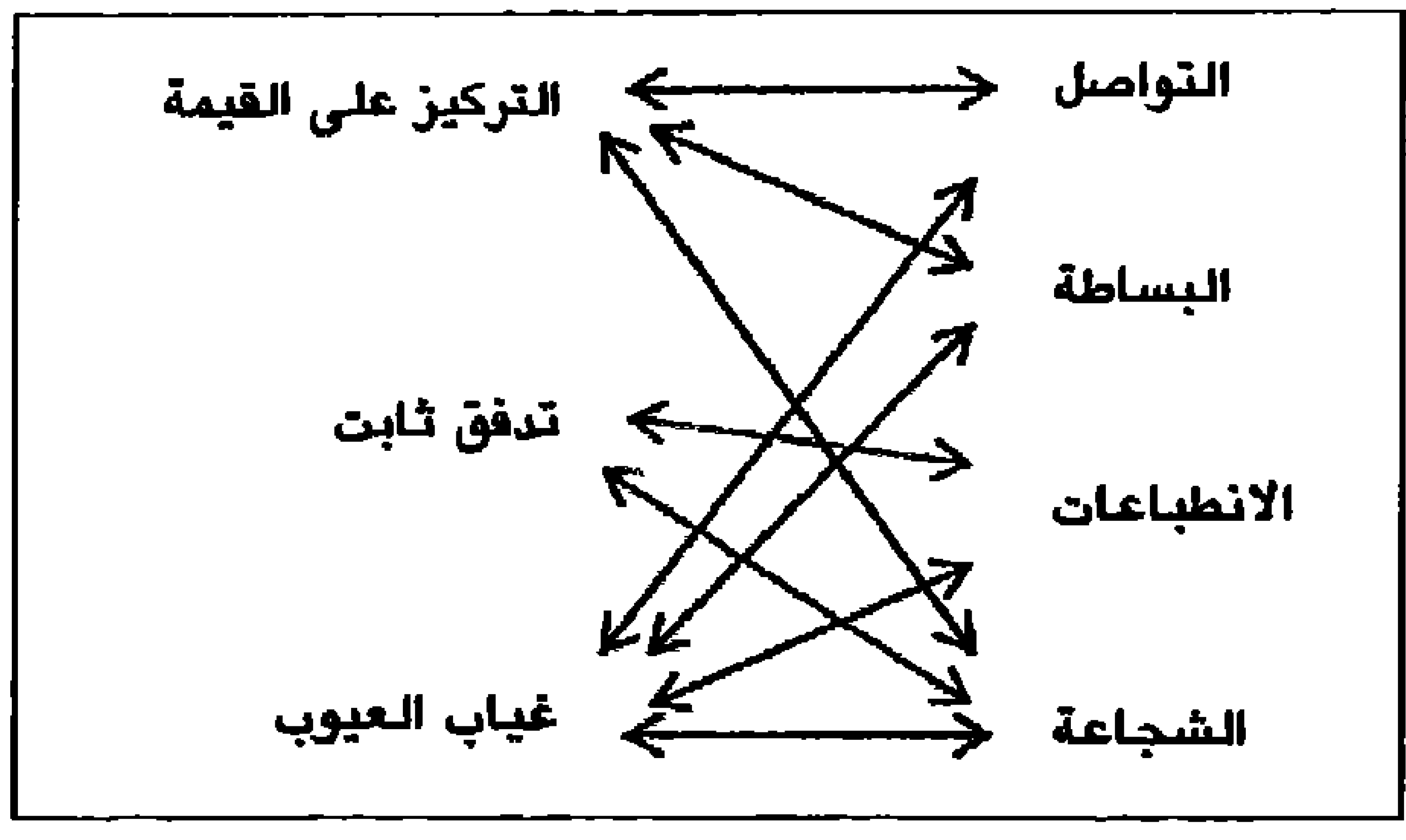
أما متحكمات منهجية XP فهي كما يأتي:

- **التركيز على القيمة**: يجب أن يركّز المطوّرون على ما يوفّر أعظم قيمة للعميل. تحدد أولويات التطوير بناءً على أولويات العميل، وليس على القضايا التقنية التي لا توفّر أي قيمة للعميل.

- **تدفق ثابت للأنشطة**: يجب أن يعمل فريق التطوير على نسق ثابت وتجنب أعباء العمل الكثير أو القليل.

- **لا عيوب**: العيوب التي قد تكون صغيرة وبسيطة اليوم تصبح كبيرة وتصعب إدارتها مستقبلاً. يجب على الفريق إصلاح جميع العيوب المعروفة والتحقق من أنها لن تظهر ثانيةً في الإصدارات المستقبلية من خلال الاختبار المؤتمت.

الروابط بين القيم والمتحكمات مبيّنة في الشكل 11 ـ 2.



الشكل (11 ـ 2): الروابط بين القيم والمتحكمات في XP

إن **التركيز على القيمة** واضح في **البساطة**. يقوم الفريق بتطوير الميزات المهمة للعميل فقط. وهذا التركيز حاضر في **التواصل** مع العميل لاستخراج المتطلبات وأولوياتها. إضافة إلى ذلك، الانطباعات وآراء العميل هي متحكم من متحكمات التطوير، بما إن العميل يحدد أولويات ما يريد إضافته و/أو تحسينه في المنتج.

إن التدفق الثابت واضح في الانطباعات، حيث يطلب المطوّرون من العملاء تحديد أولوياتهم ويفاوضون العملاء على مقدار الوظائف التي يجب تسليمها بشجاعة، ومن دون أي خوف من اتهام من جانب العميل أن المطوّرون لا يعملون بشكل كافٍ.

إن استهداف «**غياب العيوب**» يتطلب **بساطة** في التصميم لتجنب ظهور العيوب ويتطلب **الانطباعات** من الأدوات والعملاء للحد من الأخطاء الموجودة وكشف عدم التوافق مع رغبات العملاء. إضافة إلى ذلك، إن التواصل بين المطوّرين والتجرؤ على اختبار الشيفرة البرمجية ضمن ظروف قاسية يساعد بفعالية على الحد من العيوب.

من القيم والمتحكمات، تحدد منهجية XP مجموعة من الممارسات، هي كما يأتي:

- **لعبة التخطيط** (Planning Game): يجب أن يتم التخطيط من قبل المطوّر والمدراء والعميل معاً. يقوم هؤلاء الأطراف الثلاثة معاً بكتابة سيناريوهات المستخدمين للنظام؛ ثم يحدد العميل الأولويات، في حين يحدد المدير الموارد اللازمة للمشروع، يقوم المطوّرون بالإبلاغ عما يعتقدون أنه مجدٍ. يجب أن يكون التواصل بشأن الخطة نزيهاً ومنفتحاً.

- **الإصدارات القصيرة** (Short Releases): يجب أن يتواصل تطوير النظام بتوفير إضافات صغيرة ضمن إصدارات متكررة يجب أن يستخدمها العميل لتزويد انطباعات مفيدة.

- **الاستعارة** (Metaphor): يجب أن يستخدم أفراد فريق المشروع جميعاً لغة اصطلاحية مشتركة بين المطوّرين والعملاء والمدراء بحيث يتمكن المطوّرون من فهم مجال المشكلة بشكل أفضل، وبحيث يتمكن العملاء من تقدير الحلول التي يقدمها المطوّرون بصورة أفضل كذلك. يمكن بناء هذه اللغة الاصطلاحية بناءً على التناظر الوظيفي الكلي للنظام قيد الإنشاء.

● **تصميم بسيط** (Simple Design): يجب الحفاظ على بساطة النظام وتطويره تدريجياً بتطور النظام نفسه.

● **البرمجة المدفوعة بالاختبار** (Test-Driven Development): يجب أن تكتب سيناريوهات الاختبار مع العميل قبل بدء البرمجة الفعلية؛ يجب أن تغطي هذه السيناريوهات جميع جوانب ومظاهر النظام ذات العلاقة. بهذه الطريقة، تفيد سيناريوهات الاختبار كطريقة لضمان أن تحقيق المتطلبات على هيئة مواصفات رسمية لسلوك النظام.

● **تغيير تصميم البرنامج من دون التأثير في النتائج** (Refactoring): يجب أن يتم مراجعة الشيفرة البرمجية على نحو متكرر للحفاظ على بساطتها وجعلها أسهل للفهم، وتكون القيود فيها واضحة بحيث (أ) يجب أن يتحقق هذا التبسيط من خلال الاختبار أولاً، (ب) يجب أن يتم اعتماد التبسيط عند مروره خلال جميع الاختبارات الجديدة فقط، و(ج) يجب أن يتم عمل التبسيط من خلال مطوّرين اثنين (ثنائي).

● **البرمجية الثنائية** (Pair Programming): يجب أن يعمل المبرمجون في مجموعات ثنائية دائماً، حيث يعمل أحدهم على لوحة المفاتيح ويكتب الشيفرة البرمجية في حين يقترح الآخر الأفكار ويتحقق من الشيفرة البرمجية التي يتم كتابتها.

● **ملكية جماعية للشيفرة البرمجية** (Coleective Code Ownership): يجب أن يكون لكل فرد في فريق البرمجة القدرة على الوصول لأي جزء من الشيفرة البرمجية مما كتبه مبرمج آخر. كما يجب أن يكون قادراً على تعديله واعتماده في إصدار جديد، بشرط (أ) استمرار الاختبار أولاً، (ب) اعتماد الإصدارات الجديدة التي تمر من خلال الاختبارات الجديدة والقديمة فقط، و(ج) العمل في مجموعات ثنائية.

● **التكامل المستمر** (Continuous Integration): يجب أن يتم دمج الشيفرة البرمجية بصورة مستمرة لضمان أن جميع الأجزاء تتناسب مع بعضها البعض بسلاسة.

● **أربعون ساعة عمل أسبوعياً** (Forty-Hour Work Week): يجب أن يستمر المشروع على وتيرة مستدامة مع خطوط التدفق الثابت الذي تدعو إليه

منهجية Lean. لذا، قد تكون جهود العمل الكبيرة مقبولة ومحتملة لأسبوع أو اثنين على مدى المشروع، لكن يجب أن يكون توزيع الجهد سهلاً ولا يتجاوز ما يحتمله الموظف في الظروف الطبيعية، أي 40 ساعة عمل في الأسبوع.

- **تواجد العميل في موقع العمل** (On-Site Customer): يجب أن تكون وصولية العميل والمطوّرين سهلة، ويستحسن تواجدهما في الموقع نفسه إن كان ذلك متاحاً. بهذه الطريقة، سيضمن العميل أن المطوّرين يعملون بحسب الخطة وأن بإمكانهم استلام انطباعات العميل بسرعة.

- **معايير البرمجة** (Coding Standards): يجب أن تُكتب الشيفرة البرمجية بطريقة متفق عليها من قبل جميع أعضاء فريق البرمجة لتعزيز التفاهم والمشاركة بسهولة. لا يهم ما هو المعيار الذي يجب اعتماده، علماً أن وجود معيار أمر منطقي ومقبول للجميع، لكن يجب وجود معيار.

المتحكمات والقيم والممارسات مترابطة بإحكام في منهجية XP. وهذه الروابط موضحة في الجدول 11-1.

الجدول (11 ـ 1)

العلاقات بين المتحكمات والقيم والممارسات

	المتحكمات			القيم			
	التركيز على القيمة	التدفق الثابت	لا عيوب	التواصل	البساطة	الانطباعات	الشجاعة
لعبة التخطيط	✓	✓		✓		✓	✓
الإصدارات القصيرة	✓	✓		✓		✓	
الاستعارة	✓			✓		✓	
التصميم البسيط	✓		✓		✓		
الاختبار			✓		✓	✓	✓
تغيير تصميم البرنامج بدون التأثير في النتائج		✓	✓		✓		✓
البرمجة الثنائية			✓	✓	✓	✓	
الملكية الجماعية للشيفرة البرمجية	✓		✓	✓		✓	✓
التكامل المستمر	✓	✓	✓			✓	✓

يتبـــع

تابــع

أربعون ساعة عمل أسبوعياً		✓			✓		✓
تواجد العميل في موقع العمل	✓			✓		✓	
معايير البرمجة	✓		✓		✓	✓	

11 ـ 4 ـ 1 بنية فرق العمل في منهجية XP

إن حجم وبنية فريق العمل أمران مهمان لتطبيق XP بنجاح. صممت منهجية XP للعمل مع الفرق الصغيرة (من مبرمجين إلى 12 مبرمجاً) يتواجدون في غرفة واحدة. إن حجم الفريق الصغير والموقع المشترك في غاية الأهمية لتطبيق بعض الممارسات بصورة صحيحة كلعبة التخطيط والملكية الجماعية للشيفرة البرمجية وتواجد العميل في موقع العمل وغير ذلك. إذا لم تتحقق هذه المتطلبات، فقد تساعد بعض المنهجيات السريعة الأخرى كمنهجية SCRUM والمنهجية الكريستالية. . . إلخ.

يرتبط مستوى السرعة غالباً بحجم فريق البرمجة. التواصل المباشر والتوثيق المحدود أمران ممكنان في الفرق الصغيرة فقط. على العكس من ذلك، عندما يكبر الفريق، يكبر مستوى النفقات العامة أيضاً. تتضمن النفقات العامة:

- التوثيق.
- الاتصالات الوسيطة (من خلال الوسائط كالورق).

لمشاركة المعرفة وتتبع حالة المشروع، يتطلب الأمر المزيد من التوثيق لأنه لا يمكن إجراء التفاعل المباشر بين العديد من الأطراف بعد الآن (6). إضافة إلى ذلك، تزيد أهمية التوثيق ويصبح طريقة لتحسين تشارك المعرفة. في هذه الحالة، لا تكون الشيفرة البرمجية كافية، ولا يكون التواصل المباشر بين فريق التطوير والعميل ممكناً نظراً إلى حجم الفريق.

في منهجية XP، هناك ثلاثة عناصر أساسية:

1. العميل (The Customer).
2. المطوّر (The Developer).
3. المدير (The Manager).

تكون مشاركة العميل كبيرة في عملية التطوير، وغالباً ما يكون عضواً في فريق التطوير. إن وجود العميل أمر مهم في XP، وذلك أن معظم التواصل يتم وجهاً لوجه، وتكون مرحلة جمع المتطلبات موزعة خلال المشروع كله ولا تقتصر على بدايته فقط. وعليه، غالباً ما يطلب فريق التطوير من العميل الإجابة عن بعض الاستفسارات في ما يتعلق بالمتطلبات والتحقق من صحة التنفيذ.

إن تواجد العميل يقلل كمية التوثيق المطلوبة لوصف المتطلبات بالتفصيل، كما إن مساهمته المباشرة عامل أساسي لنجاح المشروع. إضافة إلى ما تقدم، يوفّر العميل رأيه للمطوّرين لتحديد أي مشكلات محتملة مبكراً في عملية التطوير ولتجنب التأثير الكبير في جدول المشروع الزمني.

أما النشاطات الرئيسة للعميل فهي كما يأتي:

- تحديد الوظائف المطلوبة من المنتج.
- تحديد أولويات هذه الوظائف.
- تحديد اختبارات القبول (بمساعدة المطوّرين).

المطوّرون في منهجية XP ليسوا الأشخاص المسؤولين عن تنفيذ المنتج فقط؛ فهم يتفاعلون مع العميل عن قرب ويوفرون البرمجية التي تعمل وفقاً لمتطلبات العميل ويقومون بجمع آراء العميل وانطباعاته ذات القيمة. لذا، لا يتطلب الأمر أن يكونوا مطوّرين جيدين قادرين على العمل ضمن فريق فحسب، بل يجب أن يكونوا قادرين على التواصل مع العميل بلغته.

يجب على المطوّرين أن يوفروا برمجية عمل ذات جودة عالية للعميل في كل فترة برمجية دورية وجميع الانطباعات ذات القيمة. هذه المنهجية قيِّمة لكلٍّ من المطوّرين والعملاء. يمكن أن يجمع المطوّرون معلومات مفيدة لتجنب تنفيذ ميزات غير مفيدة أو خاطئة، وهذا يقلّل من الوقت المستغرق لتنفيذ الميزات المفيدة؛ بإمكان العملاء اختبار المنتج البرمجي بأنفسهم بعد بضعة أسابيع من بدء المشروع.

أما النشاطات الرئيسة للمطوّر فهي كما يأتي:

- تحليل وتصميم واختبار ودمج النظام.

- تقييم صعوبة تنفيذ المتطلبات التي حددها العميل.
- العمل في فرق ثنائية؛ أي مبرمجين ولوحة مفاتيح واحدة: أحدهما يطبع والآخر يشارك في تحديد ما يتم عمله.
- مشاركة الشيفرة البرمجية مع المطوّرين الآخرين في الفريق.

في المنهجيات السريعة، يجب على المدراء إنشاء بيئة عمل وتعزيزها، حيث إنه من الممكن وجود تفاعل مثمر بين فريق التطوير والعميل. يمكن للمدراء تحقيق هذا الهدف بتحديد أفضل الأشخاص الذين سيضمهم الفريق وتعزيز التعاون والتفاوض حول عقود المشاريع مع العميل.

غالباً ما يعمل فريق العمل في المنهجيات السريعة وفقاً لعقود تتصف بنطاق عمل متغير ـ سعر متغير بدلاً من نطاق عمل محدد ـ سعر ثابت. تعتمد هذه الطريقة على قدرة المدير على تحديد العقد الذي يحقق رضا العميل ويتيح مرونة في عملية التطوير.

أما أهم الأنشطة التي يقوم بها المدير فهي كما يأتي:

- وضع بنود التعاقد والتفاوض مع العميل عن تطبيق ممارسات منهجية التطوير السريعة.
- مساعدة العملاء والمطوّرين ليصبحوا فريقاً متماسكاً.
- المساهمة في تيسير تنفيذ مهام العملاء والمطوّرين.

العامل الأساسي للنجاح في مشروع XP هو التزام أعضاء الفريق ـ وليس التزام المطوّرين فقط، لكن التزام المدراء والعملاء. يجب أن يدعم المدراء فريق التطوير وإنشاء بيئة عمل لتطبيق XP بطريقة تؤتي ثماراً. يتوجب على العملاء التواجد للإجابة عن أسئلة واستفسارات فريق التطوير وتقييم الحلول المقترحة.

11 ـ 4 ـ 2 إدارة المتطلبات في XP

المتطلبات هي الأساس لكل المنتجات البرمجية؛ إن تحديد المتطلبات وإدارتها وفهمها من أهم المشكلات التي تواجهها جميع منهجيات التطوير[17].

الجدول (11 ـ 2)

أهم أسباب فشل المشروع

المشكلة	النسبة المئوية
متطلبات غير مكتملة	13.1
مشاركة العميل ضعيفة	12.4
نقص الموارد	10.6
توقعات غير واقعية	9.9
عدم وجود دعم من الإدارة	9.3
تغيير في المتطلبات	8.7
نقص التخطيط	8.1
متطلبات غير مفيدة	7.5

حسب دراسة أجرتها مجموعة Standish[18]، خمسة من أصل ثمانية عوامل لفشل المشروع تتعلق بالمتطلبات (انظر الجدول 11 ـ 2): متطلبات غير مكتملة، ومشاركة العميل ضعيفة، وتوقعات غير واقعية، وتغيير في المتطلبات، ومتطلبات غير مفيدة.

في XP، لا تنفّذ عملية جمع المتطلبات في بداية المشروع فقط، بل هو نشاط يدوم على مدى المشروع كاملاً. عملياً، يمكن أن يغيّر العميل كلاًّ من المتطلبات وأولوياتها في كل فترة برمجية تكرارية في المشروع بعد تقييم النظام الذي سلّمه فريق التطوير في الفترة البرمجية السابقة. هذه الآراء والانطباعات المستمرة ضرورية للحفاظ على مسار المشروع وتسليم برمجية تكون قادرة على تلبية احتياجات العميل فعلياً.

تعترف منهجية XP أن تغير المتطلبات هو مشكلة ثابتة في معظم المشاريع البرمجية؛ لذا، دعم مثل هذا التغيير أمر ضمني في العملية كعامل قوة[20]. إضافة إلى ذلك، لا تحاول منهجية XP توقع التغيير أو الاحتياجات المستقبلية، بل تركز فقط على الميزات التي يدفع العميل مقابل الحصول عليها. تتجنب هذه الطريقة تطوير هيكلية عامة جداً تتطلب جهوداً إضافية[2].

لتحسين الفهم المشترك وفعالية عملية جمع المتطلبات، يستخدم فريق XP ما يعرف بسيناريوهات المستخدمين (Stories) والاستعارات (Metaphores).

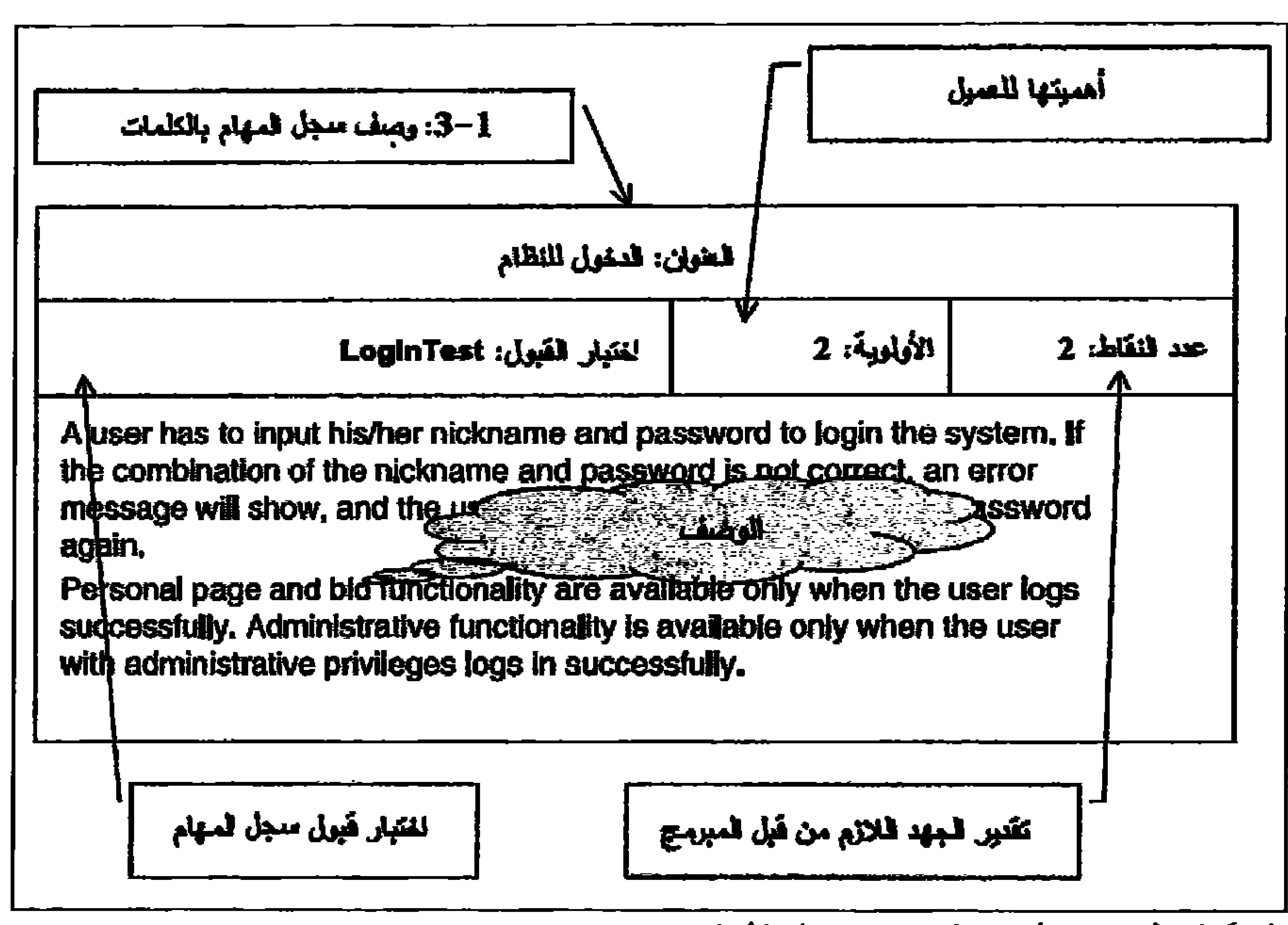
سجل مهام الاستخدام (Story) هي وصف لسلوك النظام الوظيفي. وهي

مشابهة تماماً لحالات الاستخدام، لكنها تركز على الوظائف الجزئية فقط؛ بناءً على ذلك، يمكن تقسيم حالة الاستخدام إلى العديد من سيناريوهات المستخدمين. تكتب سيناريوهات المستخدمين بلغة العميل نفسها بمساعدة المطوّرين. يحدد العميل أولويات سيناريوهات المستخدمين في حين يقيّم المطوّرون مدى صعوبتها والجهد اللازم لتنفيذها (نقاط سجل المهام).

يجب أن تكون سجلات المهام الفاعلة:

- مفهومة لكلٍّ من العميل والمطوّرين.
- مكتوبة بلغة عادية.
- مختبرة (يجب أن يكون الاختبار مكتوباً من قبل العميل بمساعدة المطوّرين).
- مستقلة عن السيناريوهات الأخرى لأكبر درجة ممكنة (على الأقل عن السيناريوهات الموضوعة في نفس الفترة البرمجية التكرارية).
- لا تتجاوز أسبوعَي عمل (أي طول الفترة البرمجية التكرارية في XP).

يبيّن الشكل 11 ـ 3 سجل مهام الاستخدام نموذجية



الشكل (11 ـ 3): تركيب سجل المهام

يعمل المطوّرون على تقييم الجهد اللازم لكل سيناريو على أساس خبراتهم وعلى أساس المشروع المستمر. قد تتطلب سيناريوهات المستخدمين الكبيرة تحقيقاً عميقاً للوصول إلى جدواها والجهد اللازم. في هذه الحالات، يطبّق المطوّرون حلول Spike التي تعتبر تحقيقاً أكثر دقة وتطبيقاً تجريبياً لهيكل الحل لإعطاء تقدير أكثر دقة.

يتم تحديد الجهد في نقاط سجل المهام التي يجب أن ترتبط بمقدار محدد من الجهد مثلاً يوم واحد لعمل زوج من المبرمجين (إذ يعمل المبرمجون دائماً في مجموعات ثنائية). إذا كان يبدو أن سجل المهام تحتاج أكثر من أسبوعين اثنين، يُطلب من العميل تقسيمها.

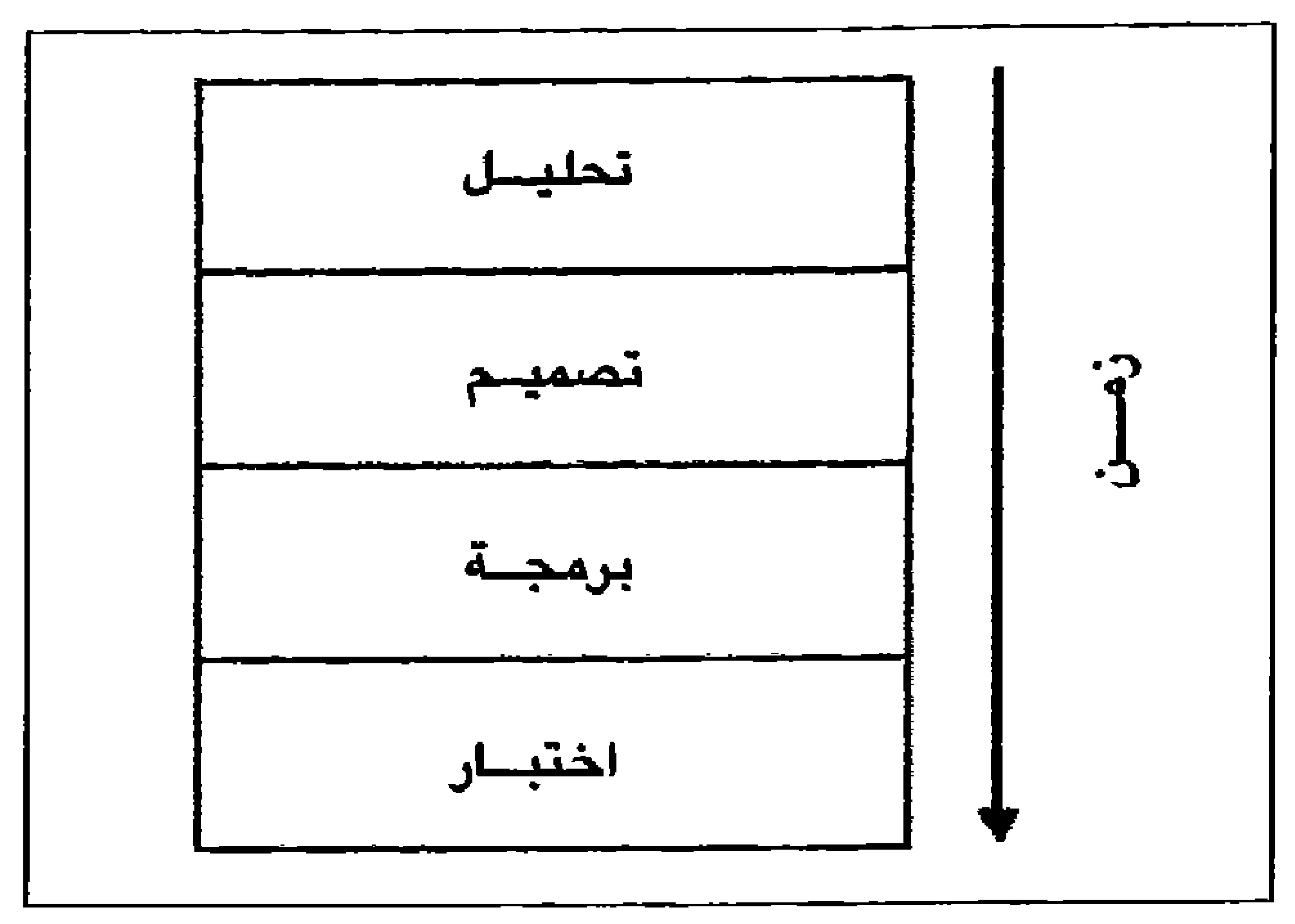
يرفق مع كل سجل مهام اختبار قبول يحدد عندما يتم تطبيق الوثيقة بصورة صحيحة ويتم قبول التطبيق من قبل العميل.

- يجب أن يكون اختبار القبول الفاعل:
- يتحقق من أن سجلات المهام قد طبقت بصورة صحيحة.
- يكون مكتوباً من قبل العميل بعد كتابة كل سجل مهام.
- يكون هناك اختبار قبول واحد على الأقل لكل سجل مهام.
- يتحقق من متى يصبح بالإمكان اعتبار سجل المهام مكتمل بحيث يتم البدء ببرمجة سجل مهام جديدة.

لكتابة سجلات مهام فاعلة، يستخدم العميل والمطوّرون الاستعارات وهي نوع من اللغة المشتركة يمكن فهمها بسهولة من الطرفين، ويجب أن تساعد هذه الاستعارات في التواصل من دون استخدام الكثير من اللغة التقنية التي قد ينتج منها سوء فهم إذا كان مجال التطبيق غير معروف لجميع أفراد الفريق.

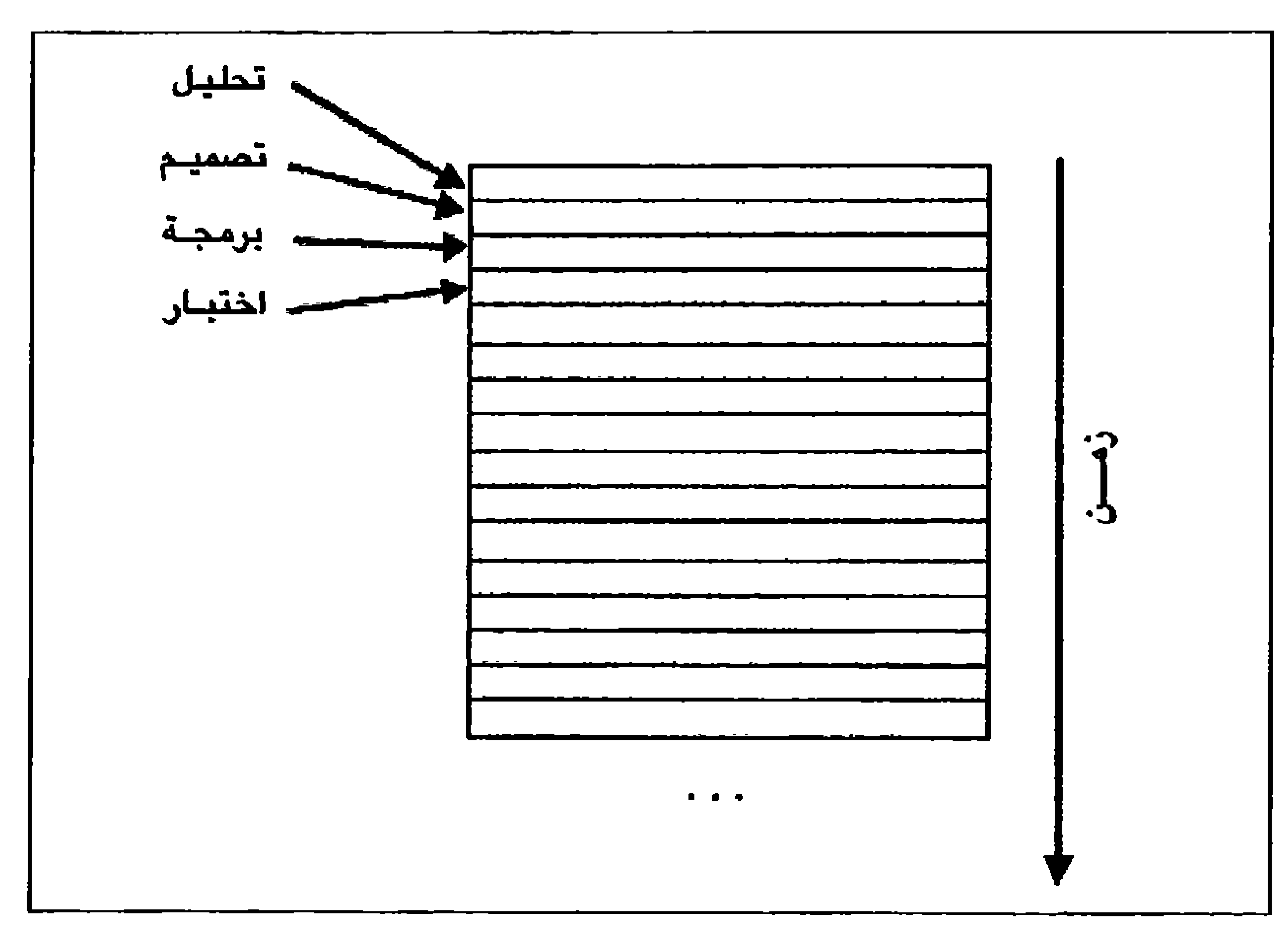
11 ـ 4 ـ 3 مقدمة لعملية التطوير بمنهجية XP

تحدد منهجيات التطوير التقليدية كمنهجية الشلال (الشكل 11-4) مراحل عملية التطوير (التحليل، والتصميم، والبرمجة، والاختبار) التي تمر عبرها مراحل الإنتاج. تم تعديل هذه الهيكلية الصارمة بطرق مختلفة بعمليات تطوير أخرى كالمنهجية الحلزونية وطرق أخرى عديدة.



الشكل (11 ـ 4) : عملية منهجية الشلال

تنظم منهجية XP عملية التطوير بطريقة مختلفة جذرياً (الشكل 11-5). إن التحديد الرسمي للمراحل لم يعد حاضراً بعد الآن، وتنظم عملية التطوير في فترات برمجية تكرارية يعمل المبرمج من خلالها على إنتاج شيء ذي قيمة بالنسبة إلى العميل. يقوم المبرمج بإجراء بعض التحليل وبعض التصميم والاختبار لكل متطلب من متطلبات العميل التي حددت في سجلات المهام بالإضافة إلى كتابة الشيفرة البرمجية.



الشكل (11 ـ 5) : عملية XP

لهذه الطريقة بعض النقاط المهمة، إذ يتم تنفيذ الاختبار قبل كتابة الشيفرة البرمجية لأن منهجية XP تستخدم طريقة الاختبار أولاً[3]: يتم كتابة سيناريوهات الاختبار قبل كتابة الشيفرة البرمجية التي تحقق تلك السيناريوهات. ليس صحيحاً أن ليس هناك تحليل أو تصميم في XP، بل إن هذه المنهجية تنشر التحليل والتصميم خلال عملية التطوير كلها ولا تركز عليهما في بداية المشروع فقط. تتيح هذه الطريقة للمبرمجين التجاوب بسرعة لطلبات التغيير والحد من هدر الموارد التي قد يتسبب عن تطبيق متطلبات خاطئة[2].

11 ـ 4 ـ 4 مقارنة منهجة XP بالمنهجيات الأخرى

يقارن الجدول 11-3 بعض المنهجيات المعروفة لدى مهندسي البرمجيات في تطوير البرمجيات التقليدية، بمبرمجي Cowboy ومبرمجي XP.

الجدول (11 ـ 3)

مقارنة XP بالمنهجيات الأخرى

مهندس برمجيات تقليدية	مبرمج Cowboy	مبرمج بحسب منهجية XP
«أحتاج تحليلاً وتصميماً كاملين قبل البدء بالبرمجة».	«لا أحتاج أي تحليل أو تصميم».	«لا أحتاج تحليلاً وتصميماً مكتملين قبل البدء بالبرمجة».
«عليّ أن أكتب جميع التوثيق بطريقة كاملة بحيث يكون الأشخاص مستقبلاً قادرين على فهم ما يجري».	«لا أحتاج أي توثيق».	«عليّ أن أكتب الشيفرة البرمجية بحيث يكون الأشخاص مستقبلاً قادرين على فهم ما يجري. أحتاج إلى كتابة بعض التوثيق الذي قد يحتاجونه فقط».
«عليّ أن أعمل بجنون لإنجاز المشروع خصوصاً عند اقتراب موعد التسليم. البرمجة عملية شاقة».	«عليّ أن أعمل بجنون لإنجاز المشروع عند اقتراب موعد التسليم فقط. البرمجة عملية ممتعة».	«يجب أن أعمل بحيث لا يزيد ذلك على 40 ساعة أسبوعياً لإنجاز المشروع خصوصاً عند اقتراب موعد التسليم، مع الحفاظ على وتيرة ثابتة وعقل متجدد. البرمجة عملية ممتعة».
«الشيفرة البرمجية هي ملكي وحدي ولا يُسمح لأي كان بلمسها!».	«الشيفرة البرمجية هي ملكي وحدي ولا يسمح لأي كان بلمسها!».	«الشيفرة البرمجية هي ملك للفريق ويسمح للجميع بتعديلها، وهي تتيح استمرارية عمليات الاختبار!».
«في النهاية، نقوم بإجراء التكامل. سيكون ذلك صعباً، لذا علينا أن نحدد بروتوكلات دمج وتكامل مُحكمة وتوثيقها بأكثر تفاصيل ممكنة».	«في النهاية، نقوم بإجراء التكامل. ما من مشكلة، الأمر سهل: سيستغرق الأمر 5».	«علينا أن نقوم بالدمج وتكامل النظام يومياً على الأقل، بحيث لا يكون هناك أي مشكلة في النهاية».

يتبع

تابــع

«يجب أن يرى العميل نسخة عاملة ونظيفة من المنتج، من الأهمية بمكان موازنة التواصل مع العميل بحيث لا يكون هناك إهدار للوقت».	«إذا كان بالإمكان، يجب أن يرى العميل نسخة نهائية من المنتج» من الأهمية بمكان «الحد ما أمكن من التواصل مع العميل بحيث لا يكون هناك إهدار للوقت».	«يجب أن يكون العميل: (أ) متفاعلاً مع المنتج الذي يتم بناؤه ومع فريق العمل، و(ب) تواجد ممثل عن العميل في الموقع إن أمكن».
«إن لم يكن معطلاً، لا تلمسه».	«حتى لو كان معطلاً، لا تلمسه! حاول إخفاءه».	«حتى لو لم يكن معطلاً، قم بتغيير تصميم البرنامج من دون التأثير في النتائج بصورة ثابتة! استخدم سيناريوهات الاختبار لضمان أنك لا تنتج عيوباً غير مرغوب فيها».
«خطط كل شيء جيداً مقدماً بحيث لا يكون هناك حاجة إلى إجراء تغيير! فالتغيير دليل واضح على التخطيط غير الكافي أو التخطيط السيئ».	«لا تخطط كل شيء، لكن حاول أن لا تجري تغييراً! فالتغيير هو دليل واضح على انزعاج العميل أو المدير».	«خطط كل شيء يمكنك توقعه وكن مستعداً للتغيير! تَحدُث التغييرات طبيعياً في المشاريع البرمجية».
«التغيير أمر سيئ! حاربه!».	«التغيير أمر سيئ! حاربه!».	«التغيير مظهر بشري! كن مستعداً للتعامل معه!».

11 ـ 4 ـ 5 آليات التحكم في منهجية XP

يتمّ ضبط أي نوع من عمليات الإنتاج من خلال آليات تحكم تحدد كيفية مزامنة الأنشطة المختلفة لتحقيق هدف مشترك.

ثمة طريقتان رئيستان للتحكم بعملية الإنتاج: تحكم خارجي وتحكم ذاتي. أما عملية التحكم الخارجي فتحدد القوانين المضافة للعملية. وهذا يعني أن العملية نفسها لا تتضمن تلك القوانين، بل إنها أضيفت لاحقاً لتنفيذ آلية تحكم. على العكس من ذلك فإن التحكم الذاتي يحدد قوانين التحكم كجزء من العملية. وهذا يعني أن العملية مصممة بحيث تكون آلية التحكم ضمنية في العملية ولا يمكن فصلها.

تستخدم أساليب هندسة البرمجيات التقليدية التحكم الخارجي أكثر. على العكس من ذلك، تستغل المنهجيات السريعة منافع السيطرة الذاتية. وهذا يعني أن العديد من ممارسات المنهجيات السريعة مصممة لإجبار المطوّرين على التنسيق من دون الطلب منهم ذلك صراحة، ما يحد من الأنشطة الإنتاجية غير المباشرة اللازمة للتنسيق فحسب. بناءً على ذلك، يمكن أن يركّز المساهمون

في المشروع على أعمالهم الجوهرية في حين يتم حل المشكلات حين ظهورها. غني عن القول إن هذا محفز واضح للجودة: إذ إن أي شيء لا يتطابق مع معايير مراقبة الجودة لا يمكن تمريره أو قبوله.

يتضمن بيان المنهجية السريعة تحكماً خارجياً في جميع مبادئه. إن التحكم الذاتي أكثر كفاءة من التحكم الخارجي بكثير؛ لكن قد يكون من الصعب تحقيقها. تحاول المنهجيات السريعة تبني طريقة تطوير تكيُّفية. يكون فيها التحكم ضمنياً بدلاً من إضافة التحكم للعملية لاحقاً. وهذه الطريقة هي إحدى معتقدات الإدارة بمنهجية Lean[22].

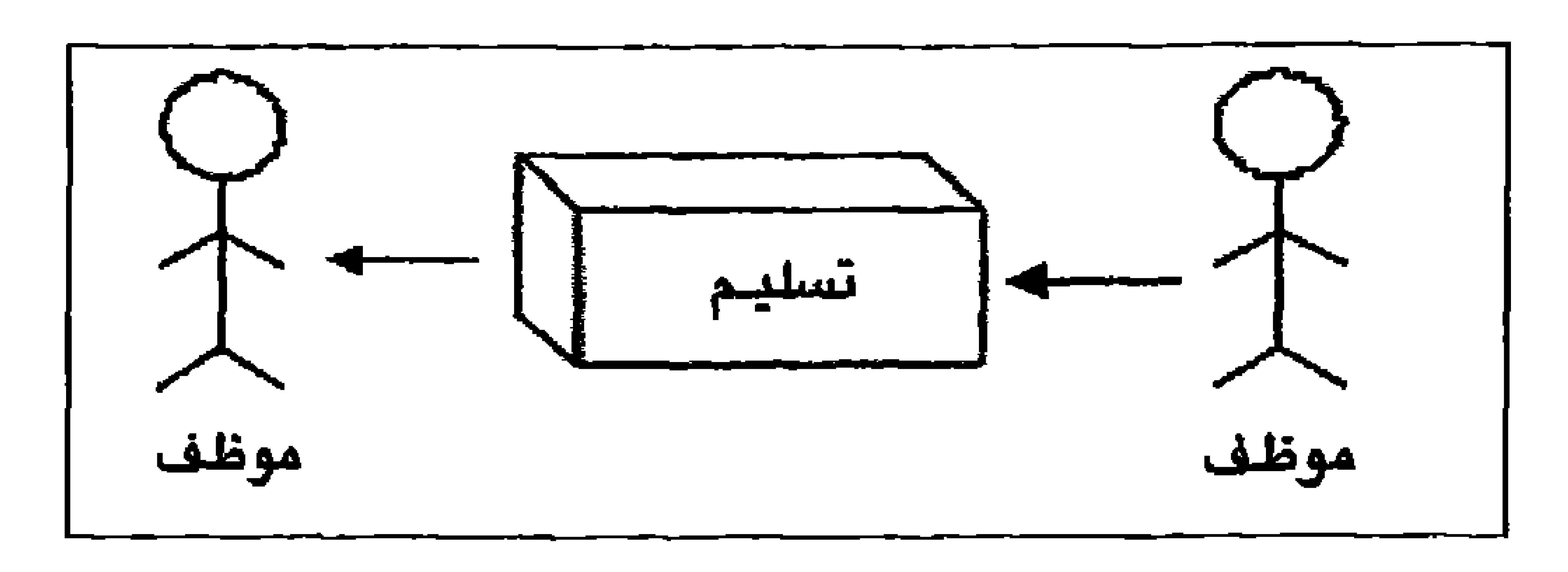
حسبما أورد كل من مالون Malone وCrownston[12] فإن الطريقة الوحيدة التي يمكن أن تكون فيها مهمتان معتمدتين على بعضهما البعض هي من خلال نوع من المصادر المشتركة. وبالتالي فإن هناك ثلاثة أنواع من الاعتمادية بين المهام، وهي:

● التحكم المتعاقب: يظهر عندما تُنشئ مهمة ما مصدراً (مخرجاً) تتطلبه مهمة أخرى كمدخل[11]. في هذه الحالة، يكون هناك تبعية تصدّرية بين المهمتين ما يتطلب ترتيب التنفيذ بصورة صحيحة[11] (الشكل 11-6).

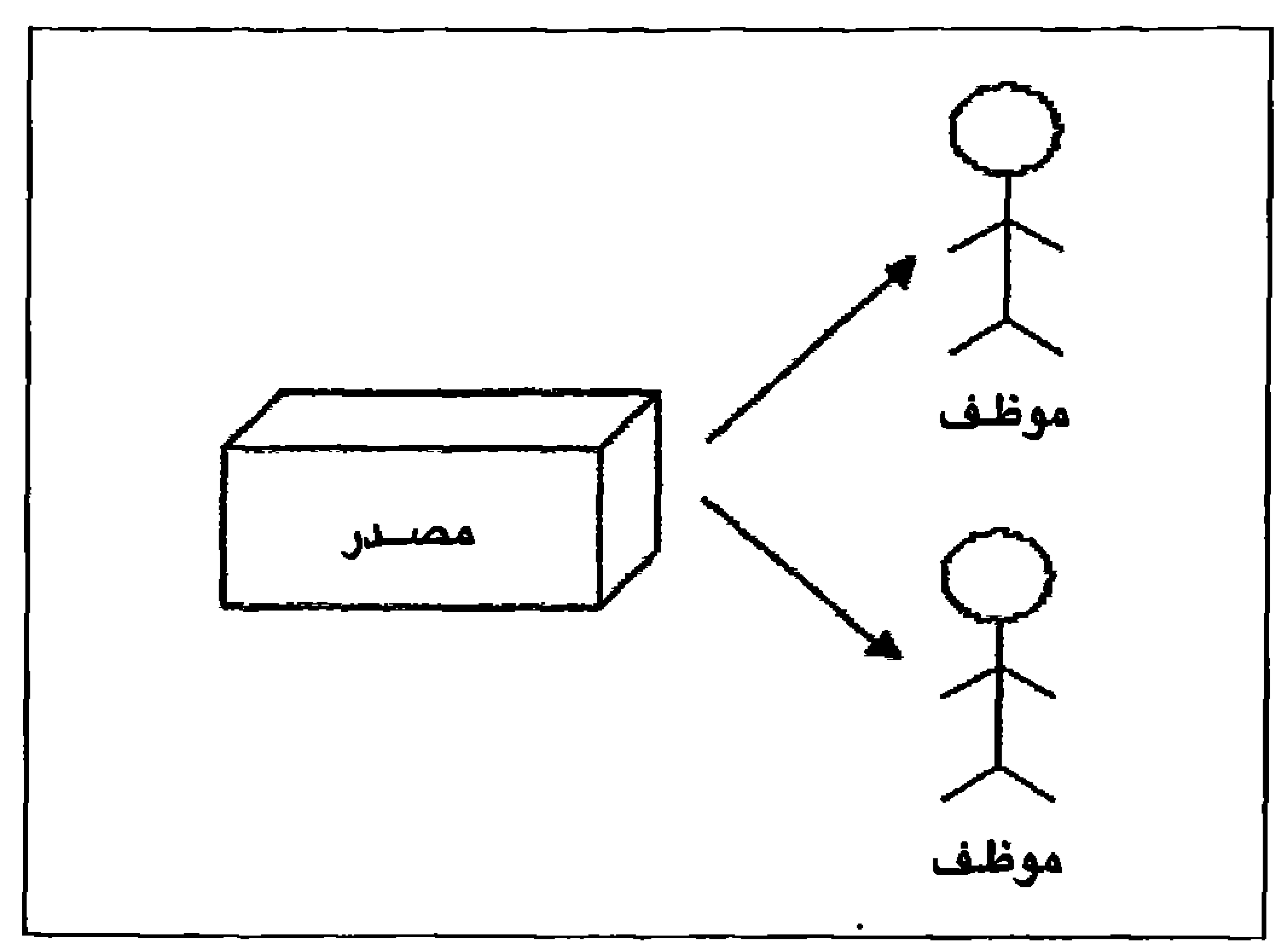
● المصادر المشتركة: تظهر عندما تشارك مهام عديدة بعض المصادر المحدودة[12] (الشكل 11-7).

● نتائج عامة: وتحدث عندما تساهم مهمتان في إنشاء النتائج نفسها (المخرجات). يمكن أن تكون هذه التبعية ذات تأثيرات إيجابية أو سلبية. في الواقع، إذا كانت كلتا المهمتين تؤديان الشيء نفسه من دون قصد، فقد ينتج من ذلك مشكلة في التكرار أو هدر المصادر. ومع ذلك، يمكن أن تؤثر المهمتان في جوانب مختلفة من المصادر المشتركة. وهذه هي الحالة في معظم المشاركين لتحقيق هدف مشترك (الشكل 11-8).

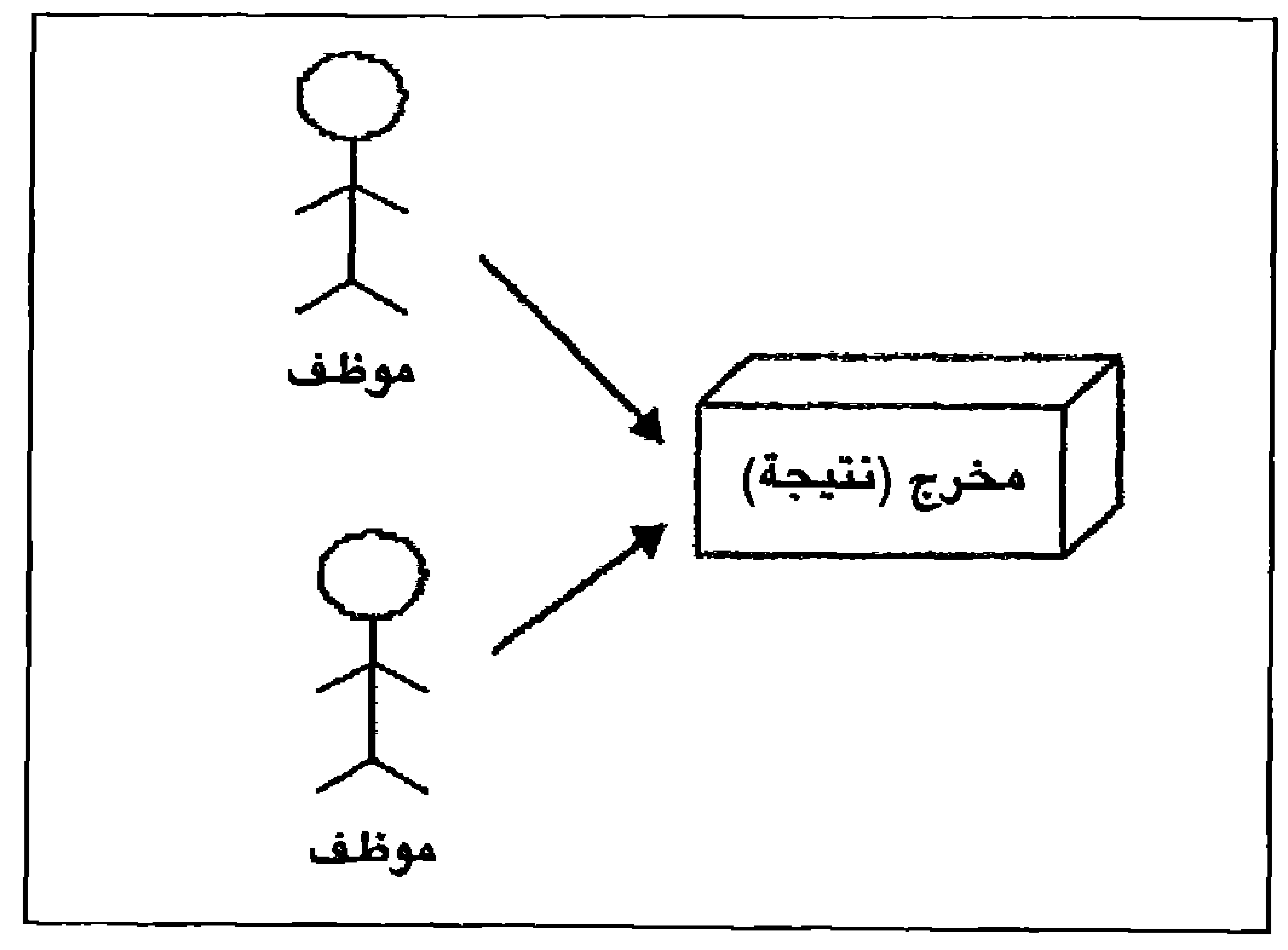
إن آلية التنسيق الأسهل هي آلية تسلسلية. في هذه الحالة، تكون الأنشطة مستقلة تماماً وترتبط فقط من خلال وثائق الإدخال والإخراج. أما آلية تنسيق المصادر المشتركة فهي أكثر تعقيداً، ذلك أنه يجب أن تحدد أولويات الأنشطة لتخصيص المصادر المشتركة للنشاط الأكثر أهمية. أما الآلية الأصعب، فهي التنسيق من خلال النتائج المشتركة. في هذه الحالة، يجب أن تعمل الأنشطة مع بعضها البعض لإنتاج النتائج نفسها.



الشكل (11 ـ 6): التحكم المتعاقب



الشكل (11 ـ 7): مصدر مشترك



الشكل (11 ـ 8): مخرج (نتيجة) مشترك

إن تقنيات هندسة البرمجيات التقليدية قائمة بمعظمها على التحكم الخارجي والتنسيق المتعاقب. على عكس ذلك، تقوم المنهجيات السريعة و XPعلى التحكم الذاتي وتنسيق المخرجات المشتركة. يسرد الجدول 11-4 آليات التحكم المستخدمة في ممارسات منهجية XP.

بالتوازي مع التحكم بالعمليات، هناك مشكلة تنظيمية، حدد Ouchi[14] ثلاث آليات كبرى تعتمد على القدرة على قياس النتائج والمعرفة المتاحة في بعض العمليات (الشكل 11-9):

- سلوكي: تستخدم هذه الآلية عندما تكون العمليات اللازمة للإنتاج نتائج معيّنة واضحة وشفافة. وهذا عمل مكتبي نموذجي.
- حصيلة: تستخدم عندما يكون بالإمكان قياس مقدار و/أو كمية النتائج. وهذه مهام احترافية أو تقنية بسيطة يمكن تنفيذها بالاستعانة بمصادر خارجية.
- زمرة: تستخدم عندما لا تكون العملية واضحة وقياس النتائج غير سهل. وهذا وضع مثالي في معظم الأعمال القائمة على المعرفة المعقدة حيث يمكن تقييم نتائج العمل النهائية لفترة محدودة فقط.

الجدول (11 ـ 4)

آليات التنسيق في منهجية XP

الممارسة	التحكم
لعبة التخطيط	ذاتي
الإصدارات القصيرة	ذاتي
الاستعارة	خارجي
التصميم البسيط	خارجي
البرمجة مع الاختبار	ذاتي
تغيير تصميم البرنامج من دون التأثير في النتائج	ذاتي
البرمجة الثنائية	ذاتي
ملكية مشتركة للشيفرة البرمجية	ذاتي
تكامل مستمر	ذاتي
أربعون ساعة عمل أسبوعياً	خارجي
تواجد العميل في موقع العمل	خارجي
معايير البرمجة	خارجي

إن فهم عملية التطوير مرتفع في تقنيات التطوير التقليدية القائمة على التخطيط المسبق حيث يكون هناك تحديد للمهام والإجراءات والأدوار. على سبيل المثال، في منهجية الشلال يكون لكل مرحلة من مراحل التطوير مخرجات واضحة وإجراء يجب لكل مرحلة اتباعه من المدخلات للنتائج (هذه قضية مختلفة تتأثر بالظروف).

فهم عمليـة التطويـر ←

القدرة على قياس النتائج ↑

النتائج	سلوكي أو نتيجـة
زمـرة	سلوكـي

الشكل (11 ـ 9): أنواع التنظيم والتحكم

تعترف المنهجيات السريعة بصعوبة فهم عملية تطوير البرمجيات وقياس النتائج. لذا فإنه يفضل الإشراف على المصادر بمنهجية الزمرة. هذا أمر واضح في بيان منهجية التطوير السريع، حيث يعتبر «التشارك» أكثر أهمية من «التفاوض» ويعتبر «التفاعل» أكثر أهمية من «العمليات».

11 ـ 5 دعم الأدوات في منهجية XP

هناك العديد من الأدوات التي تدعم التطوير باستخدام منهجية XP. لكن يفضل العديد من فرق التطوير باستخدام هذه المنهجية استخدام الأدوات التقليدية والأقل تعقيداً من الناحية التقنية كاستخدام الأوراق والأقلام والألواح والاجتماعات المباشرة حيث يلتقون وجهاً لوجه.

على سبيل المثال، يتم كتابة سجلات مهام الاستخدام على قطع صغيرة من الورق بحجم البطاقات البريدية ويثبتونها على لوح باستخدام الدبابيس. يقسم اللوح إلى ثلاثة أقسام: سجلات المهام التي يجب تنفيذها، وسجلات المهام قيد التنفيذ، وسجلات المهام المنجزة. يوفّر هذا التنسيق عرضاً مرئياً عن حالة المشروع.

حتى لو لم يكن العديد من فرق التطوير بمنهجية XP يستخدمون الأدوات البرمجية، إلا أن بعضها مفيد. منها ما هو تطبيقات معيارية غير مصممة لدعم XP، وهناك تطبيقات مصممة لغرض بعينه تم تطويرها خصيصاً لدعمها.

من بين الأدوات ذات الأغراض العامة ما يأتي:

■ أدوات النمذجة بلغة النمذجة الموحدة UML: تستخدم هذه الأدوات بطريقتين:

1. كتابة وصف التطبيق بمستوى متقدم.
2. إعادة تصميم الشيفرة البرمجية لإنشاء التوثيق.

■ أدوات التفاوض حول المتطلبات: يساعد هذا النوع من الأدوات المطوّرين والعملاء على تحديد المتطلبات وتحديد أولوياتها وإدارتها في بيئات مختلفة.

■ أدوات الرسائل الفورية: هذه الأدوات مفيدة للبقاء على تواصل مع العميل عندما لا يكون متواجداً في الموقع لمناقشة المتطلبات أو تبادل المعلومات بين المبرمجين عندما لا يستطيعون الجلوس حول مكتب واحد.

■ نظم التحكم بالإصدارات: تساعد هذه الأدوات المطوّرين على تتبع التغيير في الشيفرة البرمجية واستعادة إصدار قابل للتشغيل من الشيفرة البرمجية في حال لم تعمل التعديلات بالشكل المطلوب.

■ أدوات الاختبار المؤتمت: إن اختبار الشيفرة البرمجية باستمرار هو أحد المظاهر الرئيسة في منهجية XP. إن تنفيذ اختبار التكامل واختبار الشيفرة البرمجية بصورة متكررة أمر في غاية الأهمية لتحديد عيوب الشيفرة البرمجية ومشكلاتها حالما يتم كتابتها.

من بين التطبيقات المصممة لغرض معيّن ما يأتي:

■ أدوات إدارة المشاريع: تركز هذه الأدوات على ممارسات معيّنة من ممارسات XP، وتساعد في تخزين سجلات مهام الاستخدام واسترجاعها بطريقة إلكترونية، إضافة إلى تنظيم الفترات البرمجية التكرارية وغير ذلك.

11 ـ 6 الاستنتاجات

المنهجيات السريعة هي تنفيذ المفاهيم الأساسية للإدارة بمنهجية Lean في صناعة البرمجيات. يكون التركيز منصباً على تحسين عمليات التطوير باستمرار وتحقيق رضا العملاء، وهذان عنصران أساسيان لنجاح تلك المنهجيات. تتمثل هذه المظاهر في جميع الممارسات التي يطبقها المطوّرون يومياً.

إن هدف المنهجيات السريعة ومنهجية XP تحديداً هو تسليم برمجية ذات جودة عالية في الوقت المحدد وبحسب الموازنة المالية المرصودة، مع التركيز على ما هو قيّم ومفيد للعميل، وإنشاء قنوات تواصل لتبادل الآراء مع العميل، ودعم التغيير الذي قد يتطلبه. لكن هذه المنهجيات لا تدّعي أنها مفيدة لأي نوع من أنواع المشاريع البرمجية أو لأي نوع من أنواع التنظيمات. فالمنهجيات السريعة شأنها شأن أي تقنيات تطوير أخرى لها جوانب تنفذ بطريقة جيدة وجوانب أخرى قد تكون تقنيات أخرى أفضل منها. ومع ذلك، وبما إن هذه المنهجيات حديثة العهد فإن تحديد هذه الجوانب ما زال قائماً.

المراجع

1. P. Abrahamsson [et al.]. *Agile Software Development Methods: Review and Analysis*. Oulu, Finland: VTT Publications, 2002.

2. K. Beck. *Extreme Programming Explained: Embrace Change*. Reading, MA: Addison-Wesley, 1999.

3. K. Beck. *Test Driven Development: By Example*. Reading, MA: Addison-Wesley, 2002.

4. K. Beck. *Extreme Programming Explained: Embrace Change*. 2nd ed. Reading, MA: Addison- Wesley, 2004.

5. D. M. Berry. «The inevitable pain of software development: Why there is no silver bullet.» paper presented at: *Proceedings of the 9th International Workshop on Radical Innovations of Software and System Engineering in the Future (RISSEF 2002)*. LNCS 2941, 2004, pp. 50-74.

6. Cockburn. *Agile Software Development*. Reading, MA: Addison-Wesley, 2002.

7. D. Cohen, M. Lindvall, and P. Costa. *Agile Software Development*, DACS State-of-the-Art Report, available online at: <http://www.dacs.dtic.mil/techs/agile/agile.pdf, 2003>.

8. J. Highsmith. *Adaptive Software Development*. Dorset House Publishing, New York, 1996.

9. J. Highsmith. *Agile Software Development Ecosystem*. Reading, MA: Addison-Wesley, 2002.

10. J. Highsmith and A. Cockburn. «Agile Software Development: The Business of Innovation.» *IEEE Computer*: vol. 34, no. 9, 2001, pp. 120-127.

11. T.W. Malone and K. Crowston. «What is coordination theory and how can it help design cooperative work systems.» paper presented at: *Proceedings of the 1990 ACM Conference on Computer-Supported Cooperative Work*. Los Angeles, 1990, pp. 357-370.

12. T. W. Malone and K. Crowston. «The Interdisciplinary Theory of Coordination.» *ACM Computing Surveys*: vol. 26, no. 1, 1994, pp. 87-119.

13. T. Ohno. *Toyota Production System: Beyond Large-Scale Production*. New York: Productivity Press, 1988.

14. W. G. Ouchi. «Markets, Bureaucracies & Clans.» *Administrative Science Quarterly*: vol. 25, no. 1, 1980, pp.129-141.

15. M. Poppendieck and T. Poppendieck. *Lean Software Development: An Agile Toolkit*. Reading, MA: Addison-Wesley, 2003.

16. K. Schwaber and M. Beedle. *Agile Software Development with Scrum*. Englewood Cliffs, NJ: Prentice-Hall PTR, 2001.

17. I. Sommerville and P. Sawyer. *Requirements Engineering: A Good Practice Guide*. New York: John Wiley & Sons, 2000.

18. Standish Group, CHAOS Report 1994, available online at: <http://www.standishgroup.com/sample_research/chaos_1994_1.php>.

19. J. Stapleton. *DSDM: Dynamic System Development Method*. Reading, MA: Addison-Wesley, 1995.

20. J. E. Tomayko. «Engineering of unstable requirements using agile methods.» paper presented at: *Proceedings of the International workshop on Time-Constrained Requirements Engineering (TCRE '02)*. Essen, Germany, 2002, <http://www.enel.ucalgary.ca/tcre02/21>.

21. J. D. Thompson. *Organizations in Action: Social Science Bases of Administrative Theory*. New York: McGraw-Hill, 1967.

22. J. P. Womack, D. T. Jones. *Lean Thinking: Banish Waste and Create Wealth in Your Corporation*. New York: Free Press, 2003.

مؤلِّفو ومحرِّرو الكتاب

لورا بوكي (Laura Bocchi): باحثة مشاركة في جامعة ليستير ـ المملكة المتحدة حيث تعمل على مشروع شركة سينسوريا (هندسة برمجيات لحاسبات الطبقات الخدمية). نالت درجة الدكتوراه في علم الحاسبات من جامعة بولونيا ـ إيطاليا، حيث كانت أبحاثها لنيل هذه الدرجة العلمية متعلقة بالنماذج الرسمية التي توزع خدمات الشبكة، وركزت أيضاً على الإجراءات طويلة الأمد، وعلى بروتكولات التفاوض أيضاً.

جان بوتش (Jan Bosch): يعمل حالياً كنائب رئيس للعملية الهندسية في شركة إنتويت. تولى منصب رئيس مختبرات تكنولوجيا البرمجيات والتطبيقات في مركز أبحاث شركة نوكيا في فنلندا. وقبل التحاقه بشركة نوكيا، ترأس مجموعة بحوث هندسة البرمجيات في جامعة جرونينين في هولندا، التي يتولى فيها منصب أستاذ في هندسة البرمجيات. نال درجة ماجستير في العلوم من جامعة توينتي في هولندا ودرجة الدكتوراه من جامعة لوند في السويد. تتضمن أنشطة أبحاثه التصميم المعماري للبرمجيات وعائلات منتجات البرمجيات، إضافة إلى إدارة البرمجة القائمة على المكونات.

وهو مؤلف كتاب **«تصميم واستخدام معمارية البرمجيات: تبني وتطوير طريقة خط إنتاجي»** وحرّر عدة كتب ومجلدات أخرى، وألّف عدداً مهماً من المقالات البحثية. كان رئيس تحرير لمجلة تختص بقضايا النشر وترأس مؤتمرات عدة كرئيس عام ورئيس برنامج. خدم في العديد من لجان البرامج، ونظّم الكثير من ورش العمل.

بيرند برجيه (Bernd Brûgge): أستاذ في علم الحاسبات ورئيس اللجنة التقنية لهندسة البرمجيات التطبيقية في جامعة ميونيخ في ألمانيا، وأستاذ مساعد في علم الحاسبات في جامعة كارنيجي ميلون في بيتسبرج ـ بنسلفينيا. نال درجة الماجستير عام 1982 والدكتوراه في 1985 من كارنيجي ميلون، وحصل على الدبلوم في 1987 من جامعة هامبورغ. حصل على جائزة هربرت سيمون للتفوق في التعليم لعلم الحاسبات في عام 1995. شارك الأستاذ برجيه في تأليف كتاب **هندسة البرمجيات** كائنية التوجه بلغة النمذجة الموحدة UML وأنماط التصميم ولغة الجافا. تتضمن اهتمامات أبحاثه تطوير البرمجيات الموزعة وهندسة المتطلبات ومعماريات البرمجيات للانظمة التكيفية وعمليات تطوير البرمجيات الذكية، بالإضافة إلى تعليم هندسة البرمجيات القائمة على المشكلات.

باولو تشانكاريني (Paolo Ciancarini): أستاذ في علم الحاسبات في جامعة بولونيا في إيطاليا، حيث يقوم بالتدريس ويجري أبحاثاً في مواضيع هندسة البرمجيات. نال درجة الدكتوراه في علم الحاسبات من جامعة بيزا عام 1988. وهو عضو في جمعية مهندسي الكهرباء والإلكترونيات IEEE الجمعية العالمية في الحوسبة الآلية ACM وAICA ومنظمة ألعاب الكمبيوتر الدولية ICGA والمجموعة الخاصة للحوسبة الترفيهية IFIP SG16 التي تتمتع باهتمامات تتضمن نماذج التنسيق ولغات البرمجة أدواتية التوجه وأنظمة إدارة الوثائق التي تعمل من خلال الويب والبيئات والطرق والأدوات الخاصة بهندسة البرمجيات. وهو باحث زائر في جامعة ييل. قام بكتابة أكثر من 40 ورقة بحث نشرت في مجلات عالمية، وأكثر من 100 ورقة بحث نشرت في محاضر مؤتمرات عالمية.

أندريا دي لوشيا (Andrea De Lucia): حاصل على درجة اللوريا (درجة جامعية أولى) في علم الحاسبات من جامعة ساليرنو في إيطاليا في 1991 ودرجة الماجستير في علم الحاسبات من جامعة درام في بريطانيا في عام 1996، ودرجة الدكتوراه في الهندسة الإلكترونية وعلم الحاسبات من جامعة نابولي «فريديريكو II» في عام 1996، وكان عضواً في إدارة كلية في قسم الهندسة في جامعة سانيو في إيطاليا في الفترة بين عامي 1996 و2003. تولى أيضاً قيادة الأبحاث في مركز أبحاث تكنولوجيا البرمجيات في الفترة ما بين عامي 2001 و2003. يعمل حالياً كأستاذ دائم في هندسة البرمجيات في قسم الرياضيات وعلم الحاسبات في جامعة ساليرنو في إيطاليا، ورئيس مختبر هندسة البرمجيات ومدير للمدرسة الصيفية العالمية لهندسة البرمجيات. تتضمن أبحاثه الاهتمامات

الآتية: صيانة البرمجيات والهندسة العكسية وإعادة التصميم وهندسة البرمجيات العالمية وإدارة مسار العمل وإدارة التركيب وإدارة الوثائق وإدارة تدفق العمل وهندسة الويب واللغات المرئية والتعليم الإلكتروني. نشر الأستاذ دي لوشيا أكثر من 100 ورقة بحث عن هذه المواضيع في مجلات عالمية، وكتب كتباً وملحقات مؤتمرات. قام بتحرير كتب عالمية ومقالات خاصة في مجلات عالمية، ويساعد على تنظيم برامج اللجان لعديد من المؤتمرات العالمية في حقل هندسة البرمجيات.

مايكل فيشر (Michael Fischer): حصل على الدكتوراه في علم الحاسبات من جامعة فيينا للتكنولوجيا في النمسا عام 2007، وقد ساهم في العديد من مشاريع البرمجيات منذ عام 1985، عمل كباحث في بداية عام 2002 لمشاريع البحوث الأوروبية مُركّزاً على تحليل تطور البرمجيات لعائلات البرامج. التحق بمجموعة الأستاذ جال في جامعة زيوريخ في سويسرا عام 2005. في عام 2006 استلم منصب مهندس برمجيات في قسم علم الزلازل التابع للمؤسسة الفيدرالية السويسرية للتكنولوجيا في زيوريخ. ويساهم في مشاريع كالتحليل الزلزالي كجزء من مشروع NERIES الأوروبي والمتمثل في نظام الإنذار المبكر للزلازل أو نظام إنذار. تتناول أبحاثه الاهتمامات التالية: تطور البرمجيات واستعادة هيكلية البرنامج والهيكلية المحكومة بالنموذج.

فيلومينا فيروتشي (Filomena Ferrucci): تعمل كأستاذ مشارك في علم الحاسبات في جامعة ساليرنو في إيطاليا، حيث تقوم بتدريس محاضرات في هندسة البرمجيات وأنظمة معلومات الشبكات. وتولّت منصب مساعد رئيس برنامج في المؤتمر العالمي لهندسة البرمجيات وهندسة المعرفة الرابع عشر، وهي محررة ضيفة لموضوع مميز في المجلة العالمية لهندسة البرمجيات وهندسة المعرفة الذي تم اختياره كواحد من أفضل الأوراق العلمية التي قدمت في المؤتمر. خدمت كمساعدة رئيس برنامج للدورات التعليمية الثانية والثالثة والرابعة في المدرسة الصيفية العالمية لهندسة البرمجيات. وكانت عضوة لجنة برنامج لعدة مؤتمرات عالمية. تنصب اهتماماتها الرئيسة في أبحاثها على: مصفوفات البرمجيات لتقدير جهد البرمجة كائنية التوجه وتطبيقات الشبكة وبيئات تطوير البرمجيات واللغات المرئية والتفاعل بين الإنسان والكمبيوتر والتعليم الإلكتروني. ألفت بالتشارك أكثر من 100 ورقة بحثية نشرت في مجلات عالمية وكتب وملحقات مؤتمرات عالمية.

هارالد غال (Harald Gall): أستاذ في هندسة البرمجيات في قسم المعلوماتية في جامعة زيوريخ في سويسرا. عمل كأستاذ مشارك في الجامعة التقنية في فيينا في مجموعة الأنظمة الموزعة، وحصل من الجامعة نفسها على درجة الدكتوراه والماجستير في المعلوماتية. يهتم في بحوثه بهندسة البرمجيات بالتركيز على تطور البرمجيات وهيكلية البرمجيات وعمليات هندسة البرمجيات الموزعة والمتحركة. تولى منصب رئيس برنامج مؤتمر هندسة البرمجيات الأوروبي، وندوة الجمعية العالمية في الحوسبة الآلية (المجموعة المختصة المهتمة بهندسة البرمجيات) 2005، وورشة العمل العالمية حول فهم البرمجيات عام 2005، وورشة العمل العالمية عن تطور مبادىء البرمجيات 2004.

كارلو غيتسي (Carlo Ghezzi): أستاذ ورئيس هندسة البرمجيات في قسم الإلكترونيات والمعلوماتية في جامعة البوليتيكنيك في ميلانو في إيطاليا حيث إنه مفوض أيضاً من رئيس البحوث فيها. وهو زميل جمعية مهندسي الكهرباء والإلكترونيات IEEE الجمعية العالمية في الحوسبة الآلية ACM، وهو حائز على جائزة الخدمة المميزة للجمعية العالمية في الحوسبة الآلية (المجموعة المختصة المهتمة بهندسة البرمجيات) لعام 2006. خدم كرئيس عام ورئيس برنامج لعدة مؤتمرات مثل المؤتمر العالمي لهندسة البرمجيات، والمؤتمر الأوروبي لهندسة البرمجيات. عمل في الفترة بين عامي 2000 و2005 كرئيس تحرير لإجراءات ومنهجية هندسة البرمجيات للجمعية العالمية في الحوسبة الآلية ACM. تتمركز اهتمامات أبحاث غيتسي على هندسة البرمجيات ولغات البرمجة. وهو مهتم حالياً في المواضيع النظرية والمنهجية والتكنولوجية المتعلقة بتطوير تطبيقات شبكة انترنت عريضة. وقام بنشر أكثر من 150 ورقة بحثية وثمانية كتب.

مهدي جزائري (Mehdi Jazayeri): عميد مؤسس لكلية المعلوماتية، وأستاذ في علم الحاسبات في جامعة لوجانو في سويسرا. ويتولى أيضاً منصب رئيس النظم الموزعة في الجامعة التقنية في فيينا. أمضى سنوات عديدة في البحث والتطوير المتعلق بالبرمجيات في عدة شركات في وادي السيليكون، من ضمنها عشر سنوات في مختبرات هوليت باكرد في بالو ألتو في كاليفورنيا. ركز في أعماله الأخيرة على هندسة البرمجيات المبنية على مكونات النظم الموزعة، وبالأخص النظم المعتمدة على شبكة الإنترنت. وهو مساعد مؤلف لمفاهيم

لغات البرمجة وأساسيات هندسة البرمجيات وهيكلية البرمجة لعائلات المنتجات. وهو رفيق جمعية مهندسي الكهرباء والإلكترونيات IEEE وعضو في جمعية الكمبيوتر السويسرية والألمانية والنمساوية التابعة للجمعية العالمية في الحوسبة الآلية.

ليوناردو مارياني (Leonardo Mariani): باحث في جامعة ميلانو في إيطاليا. ينصب اهتمامه على هندسة البرمجيات، وبالأخص تحليل واختبار البرامج. قبل التحاقه بقسم المعلوماتية والأنظمة والاتصالات، كان الدكتور مارياني عالماً ضيفاً في جامعة بيدابورن. يحمل درجة اللوريا من جامعة كامرينو والدكتوراه من جامعة ميلانو، وهو عضو في جمعية مهندسي الكهرباء والإلكترونيات، وفي IEEE الجمعية العالمية في الحوسبة الآلية ACM.

سيدريك مسنيج (Ce'dric Mesnage): حصل على درجة الماجستير في علم الحاسبات من جامعة كان في فرنسا حيث اختص بالخوارزميات ونمذجة المعلومات. تناولت رسالة الدبلوما خاصته موضوع تصوّر معطيات تطور البرمجيات. وهو حالياً طالب دكتوراه في علم الحاسبات في جامعة لوغانو بإشراف الدكتور مهدي جزائري. ويعمل لحساب مشروع Nepomuk الأوروبي حيث يركز على هيكليات سطح المكتب الدلالي الاجتماعي. تنصب اهتماماته على تطور برمجيات الويب والويب الدلالي والسمات التعاونية والشبكات الاجتماعية والعلاقات ما بين علم الحاسبات والعلوم الاجتماعية.

كارلو مونتانغيرو (Carlo Montangero): أستاذ في المعلوماتية في جامعة بيزا في إيطاليا حيث يدرّس هندسة البرمجيات في قسم المعلوماتية. ساهم في أنشطة القسم كرئيس ونائب رئيس للمنهاج الدراسي لعلم الحاسبات. أمضى بضع سنوات في زيارة مراكز الأبحاث في العديد من الدول بما فيها جامعة ستانفور في بالو ألتو في كاليفورنيا وجامعة سيتي في لندن. وتنصب اهتماماته في أبحاثه على لغات البرمجة ونمذجة تطوير البرمجيات والطرق الصورية في هندسة البرمجيات. حالياً، يركز على النمذجة والتحقق من معالجة الأعمال في سياق الهيكليات خدمية التوجه وتحولات نماذج لغة النمذجة الموحدة UML لمساعدة التحقق من الهيكليات خدمية التوجه بطريقة سهلة الاستعمال.

روكو موريتي (Rocco Moretti): حصل على درجة الدكتوراه في علم

الحاسبات من جامعة بولونيا في إيطاليا. يعمل حالياً كباحث مشارك في مجال هندسة البرمجيات والهيكليات خدمية التوجه في قسم علم الحاسبات في جامعة بولونيا.

فيليبو باتشيفيتشي (Filippo Pacifici): حصل على درجة البكالوريوس العلمية والماجستير في هندسة الحاسبات من جامعة البوليتيكنيك في ميلانو عامي 2003 و2005 على التوالي ودرجة ماجستير في علم الحاسبات من جامعة الينوي في شيكاغو في عام 2006. وهو حالياً طالب دكتوراه في قسم الإلكترونيات والمعلومات في جامعة البوليتيكنيك في ميلانو حالياً. وتتضمن أبحاثه الاهتمامات الآتية: تطبيق منهجية هندسة البرمجيات لتطوير نظم حوسبة الهواتف الخلوية والمتصلة بالشبكات. وهو أيضاً عضو طالب في جمعية مهندسي الكهرباء والإلكترونيات IEEE.

ماورو بتسي (Mauro Pezze): أستاذ متخصص في هندسة البرمجيات في جامعة ميلانو في إيطاليا وأستاذ زائر في جامعة لوغانو في إيطاليا. مهتم بهندسة البرمجيات وبالذات اختبار وتحليل البرمجيات. وقبل التحاقه في قسم المعلوماتية والأنظمة والاتصالات في جامعة ميلانو بيكوكا، عمل كأستاذ مشارك في جامعة البوليتيكنك في ميلانو، وعالم زائر في جامعة كاليفورنيا - إرفين وفي جامعة ادينبرا. يحمل بتسي درجة اللوريا من جامعة بيزا ودرجة الدكتوراه من جامعة ميلانو للبوليتيكنيك. وهو عضو في الجمعية العالمية في الحوسبة الآلية ACMوفي جمعية مهندسي الكهرباء والإلكترونيات IEEE، حيث كان رئيساً للجنة الفنية المختصة بتعقيد الحوسبة (TCCX).

مارتن بنزغير (Martin Pinzger): باحث مشارك رئيس في مجموعة هندسة البرمجيات قسم المعلوماتية في جامعة زيوريخ. حصل على درجة الدكتوراه في علم الحاسبات من جامعة فيينا للتكنولوجيا في النمسا عام 2005. تنصب اهتماماته في أبحاثه على هندسة البرمجيات بالتركيز على تحليل تطور البرامج وتصميم البرامج وتحليل الجودة.

كريستيان بريهوفر (Christian Prehofer): قائد فريق أبحاث في مركز نوكيا للأبحاث. يركز في أبحاثه على الاهتمامات الآتية: الأنظمة الذاتية التنظيم والأنظمة كلية الوجود، بالإضافة إلى هيكلية البرمجية وتكنولوجيا البرمجيات لنظم اتصالات الهاتف الخلوي. تولى مناصب إدارية وبحثية مختلفة في مجال

اتصالات الهاتف الخلوي قبل التحاقه بنوكيا، وحصل على درجة الدكتوراه وتأهل لتدريس علم الحاسبات في جامعة ميونيخ التقنية في عامي 1995 و2003 على التوالي. قام بتأليف أكثر من 80 منشورة وصاحب منح 12 براءة اختراع. له دور في عدة لجان برامج لمؤتمرات عن تكنولوجيا الشبكات والبرمجيات.

فالنتينا بريسوتي (Valentina Presutti): باحثة مشاركة في مختبر علم توصيف المصطلحات (الإنتولوجي) في المجلس الوطني للأبحاث في روما. ونالت درجة الدكتوراه في علم الحاسبات من جامعة بولونيا في إيطاليا. تركز في بحوثها العلمية على تكنولوجيا الشبكات الدلالية وهندسة البرمجيات وتطبيقات تطوير النماذج للشبكة الدلالية وهندسة الانتولوجيا.

جيفري روز (Jeffrey Rose): درس علم الحاسبات في جامعة كولورادو في بولدوير قبل انتقاله إلى سويسرا لإكمال درجة الدكتوراه في جامعة لوجانو. وهو مهتم بديمقراطية المعلومات والتطور المستمر للإنترنت. يبحث حالياً في مجال صنع شبكات نظير - نظير الذكية التي تبني التواصل بين الناس والمعلومات.

البرتو سيليتي (Alberto Sillitti): مساعد أستاذ في الجامعة الحرة في بولوزانو بوزين في إيطاليا. وحصل على الدكتوراه في هندسة الكمبيوتر وهندسة الكهرباء من جامعة جانوا في إيطاليا في عام 2005 وهو مهندس محترف. يركز في أبحاثه على: هندسة البرمجيات وهندسة البرمجيات المبنية على المكونات ودمج وقياس خدمات الشبكات، وأيضاً منهجيات التطوير السريعة والبرمجة ذات المصادر المفتوحة. ويعمل في عدة مشاريع ممولة من إيطاليا والاتحاد الأوروبي في هذه المجالات.

هاري سنييد (Harry M. Sneed): حصل على درجة الماجستير في علم المعلومات من جامعة ميريلاند في عام 1969. يحتل مكانة مهمة كأحد قادة خبراء التكنولوجيا والأدوات. وهو متخصص في مساندة الشركات على تطوير برمجياتها إلى مرحلة متقدمة في التكنولوجيا. يعمل حالياً في شركة Case Consult GmbH في ألمانيا كرئيس تقني لخدمات هندسة البرمجيات. مساهماته عديدة في مجالات هندسة البرمجيات الرئيسة كالصيانة والمصفوفات وإعادة التصميم وفهم البرامج وتطوير الويب، وقد خدم كرئيس عام ورئيس برنامج وعضو لجنة تسيير وعضو لجنة برنامج لمؤتمرات مهمة في هذه المجالات.

ألف 15 كتاباً، وأكثر من 160 مقالةً تقنية. طور بنفسه أكثر من 60 أداة برمجة، وتولى دوراً في أكثر من 40 مشروعَ برمجيات. ويدرّس الآن في ست جامعات هي باسو وريغنسبورغ وكوبلينز في ألمانيا وبودابست وسيغد في هنغاريا وبينيفينتو في إيطاليا.

جانكارلو سوتشي (Giancarlo Succi): أستاذ في الجامعة الحرة في بولزانو بوزين في إيطاليا، حيث يدير مركز هندسة البرمجيات التطبيقية. تولى عدة مناصب سابقاً، منها: أستاذ في جامعة البيرتا في مدينة ادمينتون في كندا، وأستاذ مشارك في جامعة كالغاري في البيرتا، ومساعد أستاذ في جامعة ترينتو في إيطاليا. حاصل على منحة فولبرايت التعليمية. تتضمن اهتماماته في أبحاثه عدة مجالات في هندسة البرمجيات بما في ذلك البرمجة مفتوحة المصدر والمنهجيات الذكية وهندسة البرمجيات التجريبية وهندسة البرمجيات خلال الإنترنت وخطوط منتجات البرمجيات وإعادة استخدام البرامج. كتب أكثر من 150 ورقة بحثية نشرت في مجلات عالمية والكتب والمؤتمرات، وقد حرّر أربعة كتب. كان رئيساً ورئيساً مساعداً لعدة مؤتمرات وورش عمل عالمية، وهو رئيس شبكات بحثية عالمية. الأستاذ سوتشي مستشار لعدة مؤسسات خاصة وحكومية في مختلف انحاء العالم.

جيني تورتورا (Genny Tortora): أستاذة منذ عام 1990 في جامعة ساليرنو في إيطاليا حيث تدرّس نظم قواعد البيانات ومبادىء علم الحاسبات. كانت في عام 1998 عضواً مؤسساً لقسم الرياضيات وعلم الحاسبات، وعملت كرئيسة للقسم حتى تشرين الثاني/ نوفمبر عام 2000 حين تولّت عمادة كلية علوم الرياضيات والفيزياء والطبيعة. تتناول أبحاثها الاهتمامات الآتية: بيئات تطوير البرمجيات واللغات المرئية ونظم المعلومات الجغرافية ونظم المعلومات التصويرية. مؤلفة ومساعدة مؤلف لعدة أوراق بحثية نشرت في مجلات علمية وكتب وملحقات لمؤتمرات تحكيمية، وهي أيضاً محررة مساعدة لكتابين. عملت كرئيسة برنامج وعضوة لجنة برنامج لعدد من المؤتمرات العالمية. كانت عضو لجنة توجيهية في ندوة لغات وبيئات الحوسبة القائمة على العامل البشري والبشرية التابعة لجمعية مهندسي الكهرباء والإلكترونيات IEEE منذ عام 1999 حتى عام 2003. وهي أيضاً عضو متميّز في جمعية الحاسوب التابعة لجمعية مهندسي الكهرباء والإلكترونيات.

ماوريتشو توتشي (Maurizio Tucci): حصل على درجة اللوريا في علم الحاسبات من جامعة ساليرنو في إيطاليا عام 1988، ومذاك وهو يعمل كأستاذ في علم الحاسبات في جامعة ساليرنو حيث يقوم بتدريس دورات برمجة لطلاب البكالوريوس ودورات تصميم النظم التفاعلية للخريجين. تنصب اهتماماته في بحوثه على النماذج الرسمية وتقنيات تصميم وتطبيق البيئات المرئية وتقنيات التبويب المبنية على المحتوى لاسترجاع قائمة بيانات الصور وعلى أدوات هندسة البرمجيات.

جيليس فان غارب (Jilles van Gurp): مهندس أبحاث في مركز أبحاث نوكيا في هلسينكي في فنلندا، حيث تم ضمه للعمل في مشاريع أبحاث متعلقة بخدمات والبحث في الهاتف الخلوي، ويعمل حالياً على مبادىء متعلقة بتطبيقات المساحات الذكية. تتضمن اهتمامات أبحاثه أطر العمل كائنية التوجه وخطوط إنتاج البرمجيات وإدارة تغير البرمجيات تأكل تصاميم البرمجيات والحوسبة كلية الوجود. وحصل على درجة الدكتوراه من جامعة غرونيغن في هولندا عام 2003. قام بنشر عدد كبير من المقالات المتعلقة بالموضيع السابقة الذكر. في عام 2005 وقبل التحاقه بنوكيا، عمل كمدير إطلاق برامج في GX Creative Online Development وهي شركة مزودة رائدة في هولندا لأنظمة إدارة المحتوى، وعمل أيضاً كباحث في جامعة غرونيغن في معهد بلياكينيا للتكنولوجيا في السويد.

جويسبي فيساجيو (Giuseppe Visaggio): أستاذ في هندسة البرمجيات في قسم المعلوماتية في جامعة باري في إيطاليا. تغطي اهتماماته العلمية الحالية عدداً من مجالات هندسة البرمجيات وهي: جودة البرمجيات وهندسة البرمجيات التجريبية وإنتاج وصيانة البرمجيات القائم على خطوط الإنتاج وخدمات المحتوى والويب وإدارة المعرفة ومعمل الخبرة. نشر في هذه المجالات كتباً ومقالات في مجلات عالمية، وقدّم أوراقاً علمية في عدد من المؤتمرات. وهو عضو في عدة لجان برامج مؤتمرات عالمية ومراجع لعدة مجلات علمية مختصة في هندسة البرمجيات في مجال الهندسة العكسية وفهم البرمجيات وصيانتها وهندسة البرمجيات. كان عضواً في اللجنة التوجيهية للمؤتمر العالمي لصيانة البرمجيات حتى عام 2003. يتولى البروفيسور دوراً في اللجنة التوجيهية للاتحاد العالمي لصناعة اللاسلكي IWPC والمؤتمر الأوروبي لصيانة وإعادة هندسة البرمجيات CSMR ومركز أبحاث

تكنولوجيا البرمجيات RCOST. وهو أيضاً عضو في شبكة أبحاث هندسة البرمجيات العالمية (ISERN) التي تتضمن العديد من الجامعات والصناعات في كافة أنحاء العالم.

تيمو وولف (Timo Wolf): هو مساعد بحوث في الجامعة التقنية في ميونيخ حيث نال درجة الدبلوم في علم الحاسبات عام 2003. ويعمل للحصول على درجة الدكتوراه حيث يغطي بحثه المواضيع الآتية: هندسة المتطلبات والتصميم كائني التوجه ودعم الأدوات والتطوير الموزع في عدة مواقع.

ثبت المختصرات

AOSE	Agent-Oriented Software Engineering	هندسة البرمجيات ذات التوجه الأدواتي
SOA	Service-Oriented Architecture	الهيكلة ذات التوجه الخدماتي
MDA	Model-Driven Architecture	الهيكلية المعتمدة على النماذج
WSA	Web Services Architecture	هيكلية خدمات الويب
OGSA	Open Grid Service Architecture	هيكلية خدمات الشبكية المفتوحة
SOSE	Service-Oriented Software Engineering	هندسة البرمجيات ذات التوجه الخدماتي
DPS	Distributed Problem Solving	حل المشاكل الموزعة
CNP	Contract Net Protocol	بروتوكول الشبكة التعاقدي
FIPA	Foundation for Intelligent Physical Agents	مؤسسة الأدوات المادية الذكية
MDA	MODEL-DRIVEN ARCHITECTURE	الهيكلية المعتمدة على النموذج
WSRF	Web Service Resource Framework	إطار عمل موارد خدمات الويب
WSDL	Web Services Description Language	لغة وصف خدمات الويب
FSM	finite state machine	آلة تحديد الحالة المحدودة
UML	Unified Modeling Language	لغة النمذجة الموحدة
ASD	Adaptive Software Development	تطوير البرمجيات التكيفية
XP	eXtreme Programming	البرمجة القصوى
DSDM	Dynamic Systems Development Method	أسلوب تطوير النظم الديناميكية
FI	Fitness of Investment goals ()	كفاءة أهداف الاستثمار
PM	Project Management (PM)	إدارة المشاريع
MF	Multiview Framework (MF)	إطار العمل متعدد العروض
EI	empirical investigations = (EI)	الاستقصاء التجريبي

GQM	Goal Question Metrics	مقاييس السؤال المستهدف
QF	Quick Fix	إصلاح سريع
IE	Iterative Enhancements	تحسين تكراري
EM	Extraordinary Maintenance	الصيانة الاستثنائية
RE	Reverse Engineering	الهندسة العكسية
CORBA	Common Object Request Broker Architecture	هيكلية وسيط طلب الكائن المشترك
MLOC	Million Lines of Code	مليون سطر من الشيفرة البرمجية
OWL	Web Ontology Language	لغة توصيف مصطلحات الويب
RDF	Resource Description Framework	كإطار عمل وصف المصادر
WSFM	Web service Modeling Framework	إطار عملي نمذجة خدمة الويب
WSMO	Web service Modeling Ontology	توصيف نمذجة خدمة الويب
OCL	Object Constraint Language	لغة قيود الكائن
CSP	communicating sequential processes	عمليات الاتصال المتسلسلة
CCFG	Class Control Flow Graph	مخطط تدفق تحكم النوع
JIGs	Java Interclass Graphs	مخططات نوع الداخلي لجافا
WAP	Wireless Application Protocol	بروتوكول التطبيقات
CGI	(common gateway interface	واجهة البوابة المشتركة
RPC	remote procedure call	بنمط اتصال إجرائي عن بعد
WHATWG	Web Hypertext Application Technology Working Group	تطبيقات النصوص المتشعبة الخاصة بالويب
BPEL	Business Process Execution Language	بلغة تنفيذ عمليات الأعمال
UDDI	Universal Description, Discovery and Integration (UDDI	الوصف العام والاكتشاف والتكامل
WSDL	Web Service Definition Language	لغة تعريف خدمة الويب
RHDB	release history database	قاعدة بيانات الإصدارات السابقة
SV3D	Source Viewer 3D	العارض المصدري ثلاثي الأبعاد

ثبت المصطلحات

Abstract	نظري (ملخص)
Abstract Data Type	نوع البيانات الملخص
Abstractions	ملخصات
Acceptance Test	اختبار القبول
Accessability	إمكانية الوصول
Activerecord	العملية، السجل الفعال
Activity Diagrams	مخططات النشاط
Adapter Pattern	نمط الموائم
Adaptive Software Development (Asd)	تطوير البرمجيات التكيفية
Agent(S)	أداة (أدوات)
Agent-Oriented Methods	المنهجيات أدواتية التوجه
Agent-Oriented Software Engineering (Aose)	هندسة البرمجيات أدواتية التوجه
Agent-Oriented Software Life Cycle	دورة حياة البرمجيات أدواتية التوجه
Aggregation	تجميع
Agile	المنهجيات السريعة
Agile Manifesto	بيان منهجية التطوير السريع
Agile Methods	منهجيات التطوير السريعة
Agile Modelling	النمذجة السريعة
Algebraic Specifications	الخصائص الجبرية
Alternative Hypothesis	الفرضية البديلة
Analysis	تحليل
Analysis Activities	الأنشطة التحليلية

And Validation	التحقق من فاعلية
Applications	تطبيقات
Architecture Slices	الشرائح الهيكلية
Architecture(S)	هيكلية/ هيكليات
Assessment And Improvements	التقييم والتحسينات
Association	ارتباط ، اتحاد/ اتحادات
Association-End Multiplicity Coverage	تغطية نهاية الارتباط التعددية
Automatic Generation Of	الإنشاء الأوتوماتيكي
Binding	ربط
Blog Systems	نظم التدوين
Boundary Interior Loop Coverage	تغطية حلقة الحدود الداخلية
Broker	وسيط
Browser	المتصفح
Business Change	تغيير في الأعمال
Business Process Execution Language (Bpel)	لغة تنفيذ عمليات الأعمال
CASE Tool(S)	أدوات هندسة البرمجيات بمساعدة الحاسوب
Cause Construct	بناء، مُوجه
CGI (Common Gateway Interface)	واجهة البوابة المشتركة
Change Coupling	تقارن التغيير
Choreographed	مصمّم
Choreography.	التصميم
Class Attribute Coverage	تغطية خاصية النوع
Class Control Flow Graph (Ccfg)	مخطط تدفق التحكم بالنوع
Class Diagrams	مخططات النوع أو الصنف
Class(Es)	نوع/ أنواع، صنف/ أصناف
Client-Server	خادم - تابع
COCOMO Model	نموذج COCOMO
Code-Based	الاختبار المحدد بالشيفرة البرمجية
Collaboration Diagrams	مخططات التشارك
Collaborative	تعاوني
Collaborative Tagging Systems	نظام العلامات التشاركية

Collection Coverage	تغطية التحصيل
Commonality	القواسم المشتركة
Communicating Sequential Processes (Csp)	عمليات التواصل المتعاقبة
Compatibility	التوافقية
Component(S)	عنصر/ عناصر
Composition(S)	التركيب/ البنية
Compositional Product Family Approach	منهجية عائلة المنتج التركيبية
Compound	مركب
Conclusion	استنتاج
Concurrency	تزامن
Condition Coverage	تغطية الحالة
Conjecture	حدس
Construct	بناء / بنية
Consumer	المستهلك
Context-Aware	إدراك حسب السياق
Contract Net Protocol (Cnp)	بروتوكول الشبكة التعاقدي CNP
Coordination	تنسيق
Coordination Relation	علاقة تنسيقية
Corba	هيكلية وسيط طلب الكائن المشترك
Cots	البرمجيات الجاهزة التجارية
Coupling	التقارن
Coverage Criteria	معايير التغطية (الشمول)
Criteria	معايير
Crystal Family	العائلة الكريستالية
Cvs	نظام الإصدارات المتزامنة
Cyclomatic Number	العدد السيكلوماتي
Data Flow (DF)	تدفق (استمرارية) البيانات
Deployment	نشر
Design	تصميم
Design For Change	تصميم قابل ومعدّ للتغيير
Design Pattern(S)	أنماط التصميم

Design Time	فترة التصميم
Developer Contribution	مساهمة المطور
Development	تطوير
Diagrammatic Specifications	مواصفات بيانية
Discovery	اكتشاف
Dispatcher	المحول
Distributed Problem Solving (Dps)	حل المشكلة الموزعة المواقع
Distributed Systems	النظم الموزعة المواقع (المنتشرة)
Divide-And-Conquer Approach	منهجية «فرق تسد»
Django	إطار العمل Django
Dynamic	ديناميكي
Dynamic Systems Development Method (Dsdm)	منهجية تطوير النظم الديناميكية
Dynamic Type	نوع ديناميكي
Each Message On Link Coverage	تغطية كل رسالة في الرابط
E-Business	الأعمال الإلكترونية
Eclipse	نظام Eclipse
Effect Construct	بناء التأثير
Empirical Investigation(S)	الاستقصاء التجريبي
Encapsulation	إخفاء التفاصيل
Encapsulation Of	تعليب، تضمين
Endogenous Control	التحكم الذاتي
Equivalent Scenarios	سيناريوهات متماثلة
E-Science	العلوم الإلكترونية
Event(S)	حدث/ أحداث
Evolution Of Composition	تطور التركيب
Evolution	تطور
Evolutionary	تطوري
Evolutionary Aspects	مظاهر التطور
Exception Handler(S)	معالج الحالات الاستثنائية
Exception(S)	حالات استثنائية

Exogenous Control	سيطرة خارجية
Experience Factory (Ef)	مصنع التجربة
Experiment/Controlled Experiment	تجربة / تجربة متحكم بها
Experimental Subjects	الأفراد التجريبيين (المتطوعون)
Explorative Investigation	استقصاء استكشافي
External	خارجي
Extreme Programming (XP)	البرمجة القصوى (XP)
Fault(S)	خطأ (عيب)
Feature Evolution	تطور الميزات
Final	نهائي
Finite State Machine(S) (Fsms)	آلة الحالة المنتهية
Firewalls	الجدار الناري
Formal Specifications	مواصفات نظامية
Foundation For Intelligent Physical Agents (FIPA)	مؤسسة الأدوات المادية الذكية (FIPA)
Fractal	الكسيري
Framework(S)	إطار عمل / أطر عمل
Full Predicate Coverage	تغطية المسند التام
Gaia	منهجية Gaia
Generalization Coverage	تغطية التعميم
Generalization Criterion	معيار التعميم
Generic	عام/ شامل
Global Coordination Space (Gcs)	حيز تنسيق شامل
Globus Toolkit 4 (Gt4)	أداة (GT4)
Goal Question Metrics (Gqm)	مقاييس السؤال المستهدف
Graphical	واجهة رسومية
Grid	الشبكة، نقش
Grid Agents	أدوات الشبكة
Grid Services	خدمات الشبكة
Grid Services Architecture (Gsa)	هيكلية خدمات الشبكة
Gui	واجهة الاستخدام التصورية

Halstead Intelligent Content	محتوى Halstead الذكي
Halstead Mental Effort	جهد Halstead الذهني
Halstead Program Difficulty	صعوبة برنامج (Halstead)
Heuristic	حدس مهني
Hierarchy Of	التسلسل الهرمي
Html	لغة ترميز النصوص التشعبية
Https	بروتوكول نقل النصوص التشعبية المحمي
Hypertext	النص التشعبي
Hypothesis	فرضية
Identifier	معرّف
Identifier (URI)	معرّف المصادر الموحد (URI)
IEEE Standard	معايير معهد مهندسي الكهرباء والإلكترونيات
Implementation	تطبيق
In Vitro	في بيئة المختبر
In Vivo	في بيئة حيوية
Information Hiding	إخفاء المعلومات
Inheritance	التوارث
Inheritance Dependency	اعتمادية ناشئة عن التوارث
Initial	بدائي
Initialization	التهيئة
Integration	تكامل
Integration-Centric Approach	المنهجية مركزية التكامل
Integration-Oriented Approach	منهجية معتمدة على التكامل
Interclass	اختبار النوع البيني
Interface	واجهة بينية
Internal	داخلي
Interoperability	التوافقية
Intraclass	اختبار النوع الداخلي
Invariant	ثابت، لا متغير
Invocation	استدعاء
IRS-III	منصة تطوير خدمات الويب الدلالي IRS-III

IS-A Relationship	علاقة IS-A
Iterative	فترة برمجية تكرارية
J2ee	إطار عمل تطبيقات الويب بلغة الجافا (J2EE)
Java RMI	استدعاء المنهجية عن بعد الخاص بـ (Java)
Jini	تقنية شبكة (JINI) لبناء النظم الموزعة
Kiviat Diagrams	مخططات (Kiviat) البيانية
Knowledge Sources (Kss)	مصادر المعرفة
Languages	لغات
Law	قانون
Lean Management	إدارة منهجية Lean
Legacy	ميراث
Lifeline	خط العمر
Link(S)	رابط/ روابط
Localization Of	التحويل إلى المواصفات المحلية للمستخدم
Location-Aware	إدراك حسب الموقع
Maintainability	قابلية الصيانة
Maintenance	صيانة
Management	إدارة
Mccabe Cyclomatic Complexity	مقياس مكاب لدرجة تعقد النظام
Measure(S)	مقياس/ مقاييس
Mechanisms	آليات
Message Paths Coverage	تغطية مسارات الرسالة
Message(S)	رسالة/ رسائل
Metaphors	الاستعارات
Method(S)	أسلوب/ أساليب/ عمليات
Metric(S)	مقياس (معيار الأداء)
Middleware	وسيط
Migration	ترحيل
Mining	التعدين
Model Transformation	تحويل النموذج
Model View Controller (Mvc)	المتحكم بطريقة عرض النموذج

Model(S)	نموذج/ نماذج
Model-Driven Architectures	هيكليات محددة بالنموذج
Modularity	النمطية
Module(S)	وحدة/ وحدات
Multi-Agent Architectures	هيكليات متعددة الأدوات
Multiple Evolution Metrics	مقاييس التطور المتعددة
Multi-Tier	الطبقات المتعددة
Networked Systems	النظم المرتبطة بالشبكة
Nonparametric Tests	الاختبارات غير المعلمية
Null Hypothesis	فرضية العدم
Object Constraint Language (OCL)	لغة قيود الكائن (OCL)
Object Diagrams	مخططات الكائن
Object(S)	كائن/ كوائن
Object-Relational Mapping	المطابقة ذات العلاقات الكائنية
Observation(S)	ملاحظة/ ملاحظات
Observer Pattern	نمط المراقب
Of Module	واجهة الوحدة أو واجهة الموديول
Of Service	واجهة الخدمة
Off-The-Shelf	البرمجيات الجاهزة
Off-The-Shelf Components	مكونات جاهزة للاستخدام
Ontology	علم التوصيف
Open Grid Service Architecture (Ogsa)	هيكلية خدمات الشبكة المفتوحة
Open Source Software	برمجية ذات مصدر مفتوح
Operational Relationship	علاقة تشغيلية
Optimization Of Test Case Execution	الاستفادة الأمثل من تنفيذ حالة الاختبار
Oracles	المشاورات
Orchestrated	متناسق/ متناغم
Orchestration	تنسيق، مناغمة
Orchestration-Choreography Languages	معمارية متفوقة
Osgi	إطار العمل OSGI
Outsourcing	الاستعانة بمصادر خارجية

Overengineered Architecture	معمارية متفوقة
Package Diagrams	مخططات الحزمة
Package(S)	حزمة/ حزم
Pair Programming	البرمجة الثنائية
Parametric Tests	الاختبارات المعلمية
Parent	أم
Parnas Law	قانون بارناس
Parnas Theory	نظرية بارناس
Path Coverage	تغطية المسار
Peer-To-Peer	نظير ـ نظير
Php	لغة البرمجة PHP
Platform(S)	منصات البرمجيات
Polymorphism	تعدد الأشكال
Post-Condition	شرط لاحق
Postmortem Investigation	استقصاء تحليل ما بعد الحدث
Pre-Condition	شرط مسبق
Principle	مبدأ
Private	خاص
Process Calculations (I)	حسابات العملية
Process(Es)	عملية/ عمليات
Product Line Architecture	هيكلية خط الإنتاج
Program Complexity	مستوى تعقد البرنامج
Program Understanding	فهم البرنامج
Project Investigation	التحقق من المشروع
Provider	مزود
Public	عام
Publish-Subscribe	نظم نشر ـ اشتراك
Publish-Subscribe Systems	نظم نشر ـ اشتراك
Quality	الجودة
Recovering	الاسترجاع
Reengineering	إعادة التصميم

Refactoring	تغيير تصميم البرنامج دون التأثير على النتائج
Reference Architectures	هيكليات مرجعية
Reflection	الانعكاس
Regression	اختبار الانحدار
Reliability	الموثوقية/ الاعتمادية
Remote Object Coordination	تنسيق عن بعد
Remote Procedure Call (RPC)	استدعاء إجراء عن بعد
Replication	تكرار التجربة
Requirements	متطلبات
Resource Description Framework (Rdf)	إطار عمل وصف المصادر
Restoration	استرجاع
Retrospective Investigation	استقصاء بأثر رجعي
Reverse Engineering	الهندسة العكسية
Risks And Threats	المخاطر والتهديدات
Rmi Remote Method Invocation	استدعاء المنهجية عن بعد
Ruby	برمجية Ruby
Run Time	فترة التنفيذ
Scalability	التدرجية
Scrum	منهجية Scrum
Security	الأمن
Semantic Grid	الشبكة الدلالية
Semantic Web	الويب الدلالي
Semantic Web Service	خدمات الويب الدلالي
Separation of Concerns	فصل الاختصاصات
Sequence Diagrams	مخططات التتابع
Sequential	متعاقب
Service(S)	خدمة/ خدمات
Service-Oriented Architecture (Soa)	الهيكلية خدمية التوجه
Service-Oriented Architectures (SOA)	هيكليات محددة بالخدمات
Service-Oriented Computing	البرمجة خدمية التوجه
Service-Oriented Middleware	وسيط خدمي التوجه

Service-Oriented Software Development	تطوير البرمجيات خدمية التوجه
Simple Object Access Protocol (Soap)	بروتوكول وصولية الكائن البسيط
Software	برمجية
Software Architecture	هيكلية البرمجية
Software Components	مكونات البرمجية
Software Composition	تركيب البرمجية
Software Costs	تكاليف البرمجية
Software Development Process	عملية تطوير البرمجية
Software Evolution	تطور البرمجيات
Software Process	عملية برمجية
Software Product Families	عائلات المنتج البرمجية
Software Visualization	تصوّر البرمجية
Specializations	التخصصات
Specification	مواصفة
Specification-Based Techniques,	تقنيات قائمة على المواصفات
Specifications	مواصفات
Spiral	المنهجية الحلزونية
State(S)	حالة (حالات)
State-Based Behavior Of	سلوك معتمد على الحالة
Statecharts	مخططات الحالة
State-Dependent Behavior	سلوك المعتمد على الحالة
Static	ثابت (مستقر)
Static Type	نوع ثابت
Strategy Pattern	نمط الإستراتيجية
Structural	هيكلي، بنيوي
Structured Query Language (Sql)	لغة الاستعلام الموحدة
Stub(S)	شيفرة تحويل العوامل
Subclass	نوع فرعي (مشتق من رئيسي)
Subscription	إشتراك
Superclass	نوع رئيسي
Survey	مسح

Systems	نظم (أنظمة)
Tag	علامة
Tagging	وضع علامات
Temporal Logic Specifications	المواصفات المنطقية المؤقتة
Test Activities	أنشطة اختبار
Test Case(S)	سيناريو اختبار
Test Suites	مجموعات اختبار
Test-Driven Development	تطوير البرمجيات المحدد بالاختبارات
Testing	اختبار
Theory	نظرية
Time	زمن
Transition Coverage	تغطية الانتقال
Transition(S)	انتقال
Translation Time	فترة الترجمة
Tropos	منهجية Tropos
Tuple-Space	حيز – قائمة مكونات
Tuple-Space Systems	نظم حيز – قائمة مكونات
Types	أنواع
Unified Modeling Language (Uml),	لغة النمذجة الموحدة
Unified Process (Up)	العملية الموحدة
Uniform Resource	المورد الموحد
Universal Description, Discovery And Integration (UDDI)	الوصف العام والاكتشاف والتكامل (UDDI)
Url	محدد مواقع المصادر الموحدة URL
Usability	قابلية الاستخدام
Use Case	سيناريو استخدام
User Stories	سجل مهام الاستخدام
Uses Dependency	اعتمادية الاستخدام
Validity	صلاحية
Variable(S)	متغير / متغيرات
Verification	التحقق من صحة

Versioning	ترقيم الإصدارات
View(S)	طريقة عرض
Visibility	قابلية الرؤية
Visibility	وضوحية
Visualization	التصوّر
Waterfall	منهجية الشلال
Web	الويب
Web Application Framework(S),	أطر عمل تطبيقات الويب
Web Application(S)	تطبيقات الويب
Web Engineering	هندسة الويب
Web Ontology Language (Owl)	لغة توصيف الويب
Web Service Architecture (Wsa)	هيكلية خدمات الويب
Web Service Definition Language (Wsdl)	لغة تعريف خدمة الويب
Web Service Modeling Framework (Wsfm)	أطار عمل نمذجة خدمة الويب
Web Service Modeling Ontology (Wsmo)	توصيف نمذجة خدمات الويب
Web Service Modeling Ontology (Wsmo)	توصيف نمذجة خدمة الويب
Web Service Resource Framework (Wsrf)	إطار عمل مصدر خدمة الويب
Web Service(S)	خدمات الويب
Web Services Description Language (Wsdl)	لغة وصف خدمات الويب
Wide-Mouthed Frog Protocol	برتوكول «الضفدع ذو الفم الواسع»
Wiki Systems	نظم الويكي
Workflow Management System (Wfms)	نظام إدارة سير العمل
Wrapper	منفذ الطي
Wrapping	الطي
Xml	لغة الترميز القابلة للامتداد
XP Development Process	عملية التطوير بمنهجية XP

فهـــرس

- ت -

- ث -

- ج -

- ح -

- خ -

- ن -

المنهجيات والتقنيّات وإدارة العمليات الحديثة في هندسة البرمجيّات(*)

السلسلة: تضم هذه السلسلة ترجمة لأحدث الكتب عن التقنيات التي يحتاج إليها الوطن العربي في البحث والتطوير ونقل المعرفة إلى القارئ العربي.

الكتاب: المرجع الوحيد الذي يقدم للتقنيات المستحدثة في هندسة البرمجيّات وهو يتطرق بلغة سلسلة ويسيرة إلى أربعة حقول رئيسة في هذا المضمار هي: معمارية البرمجيّات، وطرائق تأثير البرمجيّات في الحوسبة الخدماتية وفحص الأشياء والـ UML وتطوير الشبكة الحديثة، وتقنيات التطوير، وأخيراً إدارة العمليات حيث يضع أسساً لطرائق ذكية وسلسلة لعملية الإدارة.

يمثل كتابنا الحالي مرجعاً في الخطوة الأولى لممارسي هندسة البرمجيّات ومختصيها كما يعد كتاباً مثالياً لطلاب الدراسات الجامعية والعليا على حد سواء.

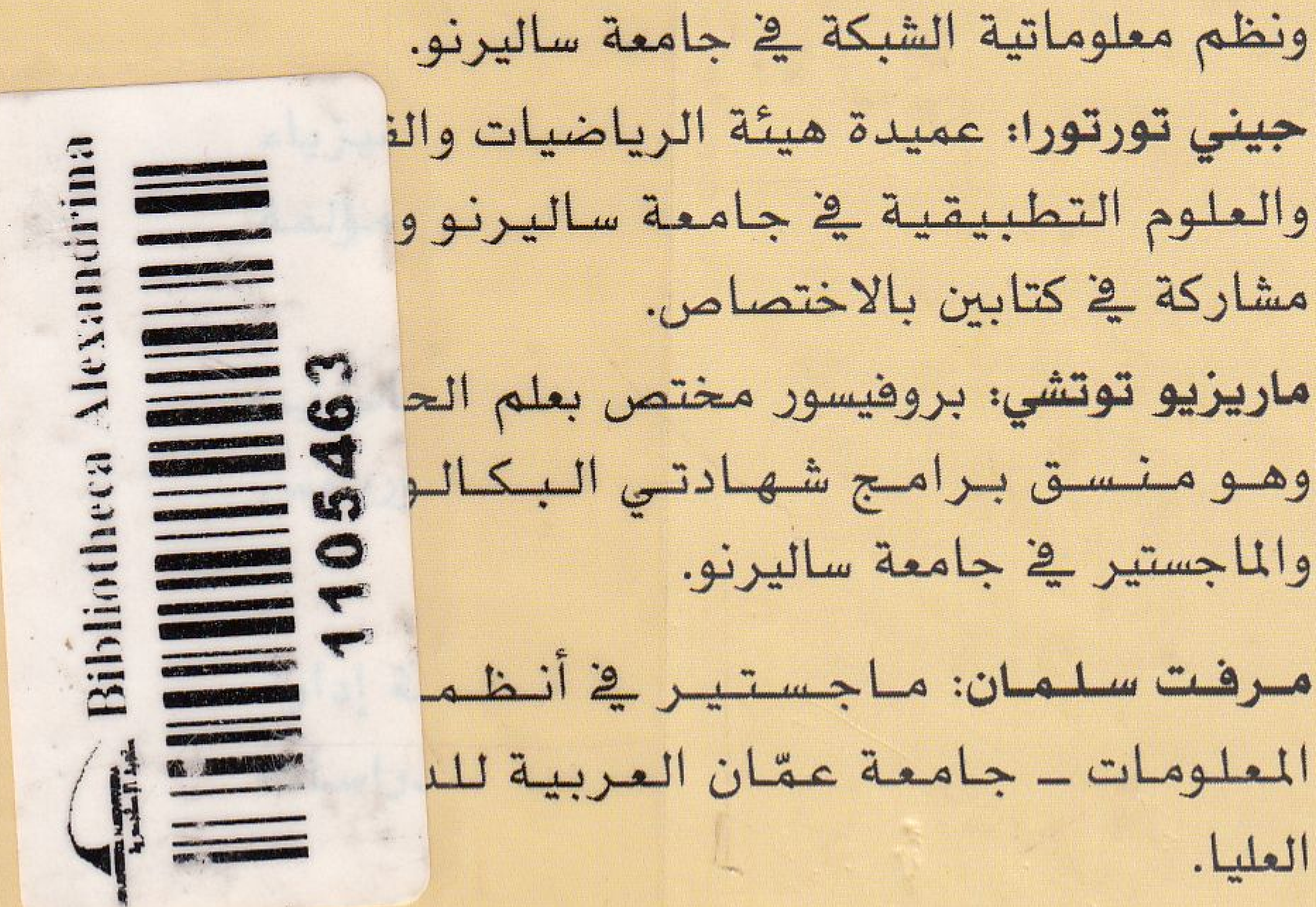
المؤلف: **أندريا دي لوتشيا:** أستاذ علم الحاسوب ومدير المدرسة الدولية الخاصة بهندسة البرمجيات في جامعة ساليرنو (إيطاليا).

فيلومينا فيروتشي: أستاذ مساعد علم الحاسوب ومعاونة مدير المدرسة الدولية الخاصة بهندسة البرمجيات، ومدرسة مادتي هندسة البرمجيات ونظم معلوماتية الشبكة في جامعة ساليرنو.

جيني تورتورا: عميدة هيئة الرياضيات وال... والعلوم التطبيقية في جامعة ساليرنو و... مشاركة في كتابين بالاختصاص.

ماريزيو توتشي: بروفيسور مختص بعلم الح... وهو منسق برامج شهادتي البكال... والماجستير في جامعة ساليرنو.

المترجمة: **مرفت سلمان:** ماجستير في أنظم... المعلومات ـ جامعة عمّان العربية للد... العليا.



Emerging Methods, Technologies, and Process Management in Software Engineering

Edited by Andrea De Lucia, Filomena Ferrucci, Genny Tortora, Maurizio Tucci

(*) الكتاب الثاني من تقنية المعلومات

سلسلة كتب التقنيات الاستراتيجية والمتقدّمة

1. المياه
2. البترول والغاز
3. البتروكيمياء
4. النانو
5. التقنية الحيوية
6. تقنية المعلومات
7. الإلكترونيات والاتصالات والضوئيات
8. الفضاء والطيران
9. الطاقة
10. المواد المتقدمة
11. البيئة

المنظمة العربية للترجمة

مدينة الملك عبدالعزيز للعلوم والتقنية KACST

الثمن: 20 دولاراً أو ما يعادلها

ISBN 978-9953-82-384-3
9 789953 823843